FH JOANNEUM

Bachelor's Thesis

# Development of an Airfoil Shape Optimization Algorithm

| | |
|---|---|
| Submitted by: | Michael Fürweger |
| Registration Number: | 1010587010 |
| Supervisor: | Dr. Wolfgang Hassler |
| Date of Submission: | $31^{st}$, January 2013 |

## Affidavit

I hereby affirm in lieu of an oath that the present bachelor's thesis entitled

**"Development of an Airfoil Shape Optimization Algorithm"**

has been written by myself without the use of any other resources than those indicated and quoted.

Graz, $31^{st}$ January 2013

Michael Fürweger

# Preface

This project is my fifth semester project and also the first part of my Bachelor's thesis for the Bachelor degree of the Luftfahrt/Aviation studies at FH Joanneum in Graz, Austria. The topic was chosen out of own interest and guided by my academic supervisor.

At this point I would like to express my gratitude to Mr. Hassler, my academic supervisor, for giving me the opportunity to work on a topic of my own interest, for advising and assisting me throughout this project.

Dedicated to my love and my family.

# Contents

# Abstract

The present thesis gives interested readers an introduction into numerical optimization. The methods introduced in the following sections can be applied generally to all kinds of optimization problems. For the purpose of this project these methods were applied to airfoil optimization in terms of drag minimization.

Within the next pages one can find methods for nonlinear, constrained and unconstrained optimization. This thesis presents the basic ideas that are necessary to understand the working principles of all methods; for detailed in-depth-information and proofs the reader may follow the references within the specific chapters.

The author has chosen a quasi-Newton method with the so-called BFGS-update for Hessian approximation. The airfoil is represented by cubical splines with the spline coordinates as optimization parameters. The optimization is constrained by a penalty method. These methods are implemented in MATLAB (R2011a) and the aerodynamic properties of the airfoil are calculated with XFOIL (v9.96). Further, an interface between MATLAB and XFOIL was necessary but creating this interface was not task of this project. Therefore an existing interface downloaded from the MATLAB file-exchange server was used.

The result of this project is an algorithm which allows to execute efficient optimization of airfoils for up to 20 spline coordinates. The user will get introduced into the necessary handles in the results section where some step-by-step examples will be explained and executed. Finally, the results will be critically examined before an outlook for future studies is given.

# Kurzfassung

Die folgende Arbeit soll dem Leser einen guten Überblick über die notwendigen Methoden numerischer Optimierung geben. Das riesige mathematische Gebiet der Optimierung wird dabei lediglich auf jene Methoden beschränkt, welche auch im Zuge der geometrischen Profiloptimierung hinsichtlich einer Widerstandsminimierung verwendet werden. Ziel dieser Arbeit ist es nicht, mathematische Hintergründe oder Beweisführungen zu durchleuchten, sondern das Verstehen der Vorgehensweise der einzelnen Methoden zu fördern. Für tiefgründigere Hintergründe sei jeweils auf entsprechende Literatur verwiesen.

Dem Leser werden die nötigen Methodiken zur nichtlinearen beschränkten und unbeschränkten Optimierung vorgestellt. Der Autor entschied sich für eine quasi-Newton Methode mit BFGS-Update zur Approximation der Hesse Matrix. Eine globale Schrittweitenstrategie wird in Form der Wolfe-Powell Methode eingeführt und beschränkt wird die Optimierung durch eine Penalty Methode. Die Flügelprofile werden durch kubische Splines repräsentiert, wobei die Spline Koordinaten den zu optimierenden Variablen entsprechen.

Das Ergebnis dieser Arbeit ist ein auf MATLAB (R2011a) und XFOIL (v 9.96) gestütztes Programm, welches effiziente Widerstandsoptimierung mit bis zu 20 Spline Koordinaten erlaubt. Als Interface wurde ein frei verfügbares MATLAB Script verwendet, welches am MATLAB File-Exchange-Server heruntergeladen werden kann. Der Leser wird anhand von Erklärungsbeispielen Schritt für Schritt in die Bedienung des Programms eingeführt, sodass später eigene Optimierungen durchgeführt werden können. Schlussendlich wird das Ergebnis der Beispieloptimierung diskutiert und eine Vorschau für eventuelle zukünftige Studien gegeben.

# List of Figures

# List of Symbols

$\alpha$ ... angle of attack
$\varepsilon$ ... finite stepwidth
$\lambda$ ... stepwidth
$\mu$ ... factor for ensuring positive definiteness
$\nu$ ... kinematic viscosity
$\varphi(t)$ ... value of approximated objective function
$\phi(t)$ ... penalty function
$\psi(t)$ ... simplified Wolfe-Powell condition

$a_i, b_i, c_i, d_i$ ... coefficients of cubical spline polynomial
$B$ ... Hessian matrix (approximated)
$C_D$ ... drag coefficient
$C_L$ ... lift coefficient
$C_L^*$ ... design lift coefficient
$C_M$ ... moment coefficient
$c$ ... chord length
$c_1, c_2$ ... user defined constants for stepwidth function
$\vec{d}$ ... direction of descent
$\vec{g}$ ... gradient
$H$ ... Hessian matrix (exact)
$I$ ... unity matrix
$M$ ... Mach number
$m$ ... second order Tailor approximation
$P_i$ ... cubical polynomial
$Re$ ... Reynolds number
$\vec{p}$ ... direction of descent
$s$ ... change in objective function
$s.t.$ ... subject to
$V$ ... velocity
$\vec{X}$ ... design vector
$y$ ... change in gradient

# 1 Introduction

Airfoil optimization is the process of deforming the shape of existing airfoils in order to get better aerodynamical characteristics in terms of drag, lift and moments. Before the advent of Computational Fluid Dynamics (CFD) trial and error was more or less the only opportunity to enhance airfoil characteristics. There are analytical aerodynamical methods described by Anderson (2011) which allow calculation of "global" change in airfoil shape like camber and thickness but only CFD makes it possible to incorporate local impacts on the airfoil surface.

Why would one want to impact beautiful, smooth airfoil surface? Well, Figure 1 shows that omnipresent standard-profiles are not always the best solution in terms of aerodynamic efficiency. Figure 1 shows the drag coefficient for a standard profile compared to an optimized version of the airfoil. By keeping in mind that the drag coefficient goes linearly with absolute drag, one can get an idea of possible improvements in terms of efficiency.

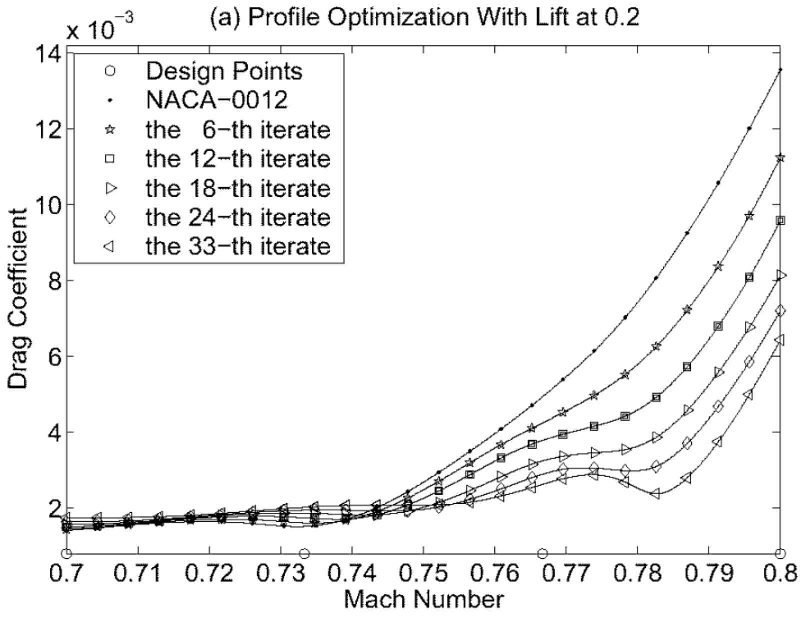


Figure 1: Drag coefficient over Mach number of original NACA 0012 profile compared to an optimized version after several optimization iterations (Li et al., 2001, p. 16)

In the early 30's of the 20th century the National Advisory Committee for Aeronautics (NACA) developed the NACA profile series, a series of profiles whose geometry is defined by 4 up to 6 designators (e.g. NACA 2412). This was done by performing an endless amount of wind-tunnel tests. Nowadays one can easily design airfoils with commercially available computers. Because of that it is not far-fetched that airfoil design is not a matter of wind-tunnel testing any more but a task for CFD-specialists.

In terms of efficiency of airplanes but also of wind turbines, engine fans and other similar applications it is very important to design profiles tailored exactly to the purpose of use. Nowadays most modern aircraft have their own tailored airfoil- and wing-shapes.

There are basically two ways of accessing the tailoring of airfoils: First it is possible to determine the necessary change of geometry of an airfoil in order to improve its characteristics. The second approach is to prescribe a certain pressure distribution and by doing so one can derive the shape of an airfoil. In aerodynamics the second approach is called *inverse airfoil design* and the basic aerodynamic methods (e.g. thin airfoil theory) are well explained by Anderson (2011), whereas Selig et al. (1992) also deal with numerical aspects.

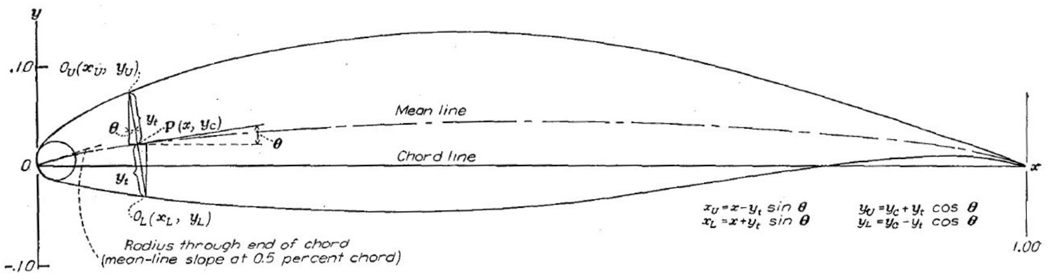
This project aimed at the first approach, to calculate the necessary deformation to achieve better characteristics. So the goal of the project was to implement an algorithm which automatically finds the most suitable shape of an airfoil for a certain flow condition in terms of a minimized drag coefficient. This problem is called *optimization problem* and mathematically speaking the goal is to minimize an objective function (e.g. drag coefficient as a function of shape). There are several different optimization methods described in literature.

The following report will contain a method section where the *quasi-Newton optimization method* with all other necessary tools will be described theoretically. It is highly recommended to study these before the program is used. Finally also some optimization results with stepwise explanations are presented in Section 3.

# 2 Methods

## 2.1 Airfoils and Geometric Representation

The standard NACA profiles have already been mentioned in the introduction. The geometry of these profiles can be described with analytical formulae. As shown in Figure 2 the idea is to apply a certain thickness distribution to a certain curved camber line, called mean line in Figure 2.



$y$
.10
$O_U(x_U, y_U)$
$\theta$
$y_t$
$P(x, y_c)$
$\theta$
Mean line
Chord line
0
$y_t$
$O_L(x_L, y_L)$
Radius through end of chord
(mean-line slope at 0.5 percent chord)
-.10
$x_U = x - y_t \sin\theta$
$x_L = x + y_t \sin\theta$
$y_U = y_c + y_t \cos\theta$
$y_L = y_c - y_t \cos\theta$
$x$
1.00

Figure 2: NACA profile with chord line and mean/camber line (Abbott et al., 1945, p.4)

The idea of the NACA series was that each series is designated by 4 to 6 numbers and these numbers define the airfoil shape (Anderson (2011) gives a detailed explanation in the NACA numbering system). One could now easily change from one standard profile to another by increasing or decreasing chamber or thickness but in the end this gives not as many freedoms as there would probably be necessary to fully optimize an airfoil.

So what kind of deformation is necessary then? It should be possible to achieve a more local deformation as it is possible with global change of camber. Figure 3 depicts global change in thickness and chamber whereas Figure 4 depicts local change of geometry. Global change would be rather too inflexible and would not bring up anything new.

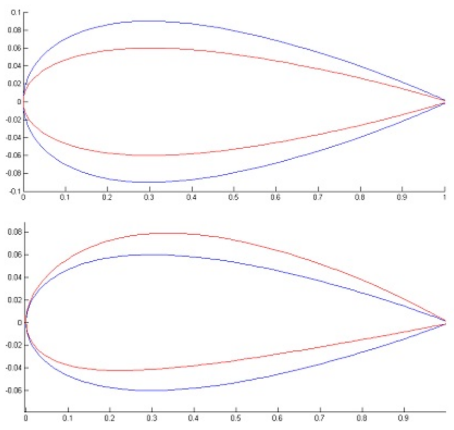


Figure 3: Global change of geometry

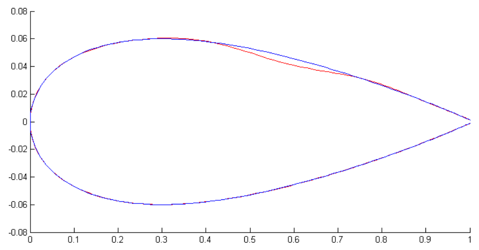


Figure 4: Local change of geometry

Therefore it is necessary to leave behind the idea of closed analytical description of airfoils and come up with a discrete description of the surface. In order to ensure a smooth surface (in contrast to that depicted in Figure 5) many coordinates are necessary. But on the other hand for optimization each coordinate means another degree of freedom (DOF) of the objective function. In other words, it would not be possible to execute optimization with as many as 150 or even more DOF. Therefore the number of optimization parameters has to be reduced drastically. On the other hand the discrete geometry should still be representative for the original profile. This point leads to a hybrid model between analytical formulation of shape and discrete shape. Interpolation is the method that defines a curve or geometry between certain coordinates with analytical formulae. This method should allow reducing the number of coordinates for optimization purposes but still deliver a representative and steady geometry.

The first step is to transform the geometry into discrete formulation by calculating the coordinates. From these coordinates a smaller number of coordinates, the so called spline coordinates (approximately 10-25 for this purpose), is extracted and the shape between them is approximated by interpolating these coordinates. There are several different methods for interpolation (e.g. Figure 5 depicts linear interpolation). For this purpose the method of cubical splines was chosen because cubical splines ensure a smooth, steady and twice differentiable contour. This method requires a closer look and will be described within the next chapter.

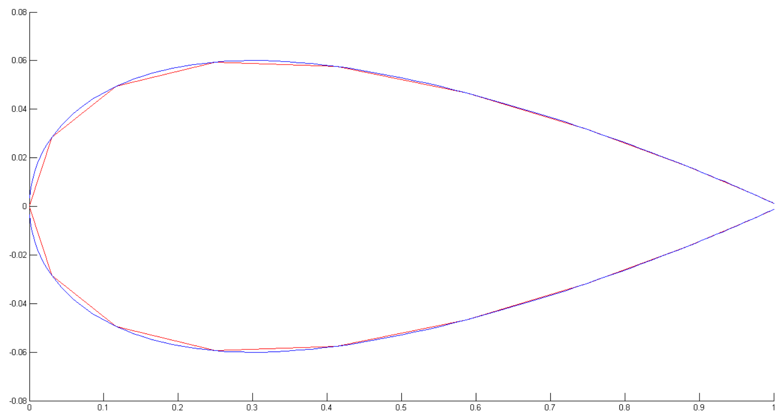


Figure 5: Smooth NACA 0012 airfoil shape vs. discrete version using 19 points and linear interpolation

### 2.1.1 Cubical Splines

The idea of cubical splines (or just splines) is to set up a system of third-order polynomials with one polynomial $P_i(x)$ valid only between two certain coordinates $x_i$ and $x_{i+1}$. There is no longer one single polynomial valid for all coordinates but $n-1$ splines for $n$ coordinates. Separation of geometry brings the advantage that the order of interpolating polynomials can be reduced. In order to interpolate more coordinates at once, higher order polynomials were necessary which tend to undulations as Figure 6 shows.

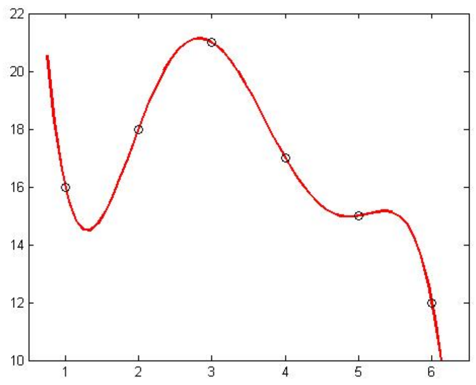


Figure 6: Higher order polynomials tend to oscillation

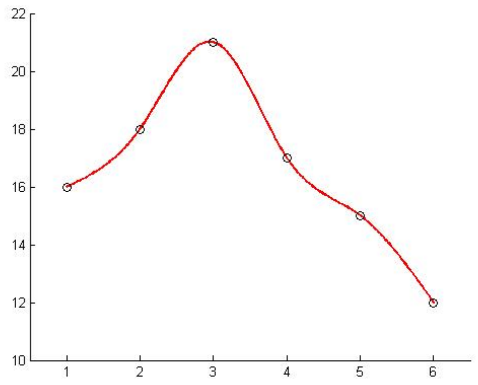
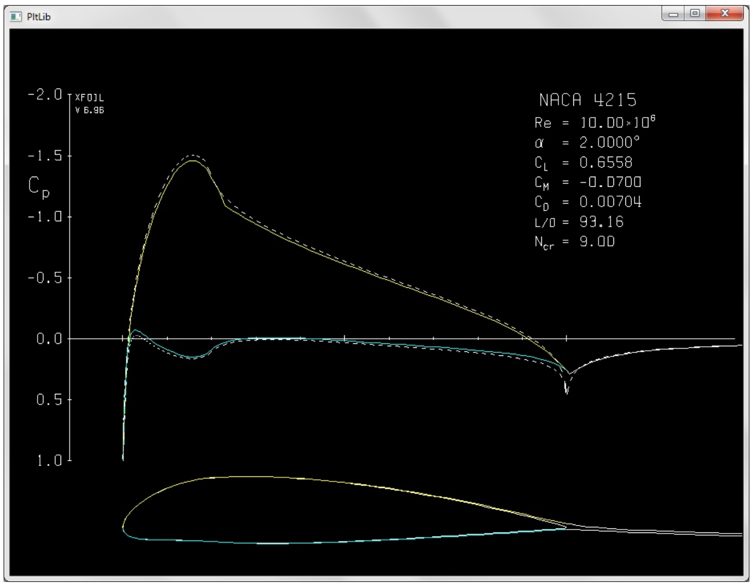


Figure 7: Interpolation solved with cubical splines

It is not that easy to calculate these splines but they have the advantage of creating a smooth and continuous surface which is twice differentiable. It is noted here that

NACA 4-series had an unsteady curvature at the point of maximum chamber and this leads to pressure peaks around this area (see Figure 8). Later on – in the 5- and 6-series – NACA increased the order of the polynomials for the mean-camber line so that this effect did not occur any longer.



NACA 4215
Re = $10.00 \times 10^6$
$\alpha$ = 2.0000°
$C_L$ = 0.6558
$C_M$ = -0.0700
$C_D$ = 0.00704
L/D = 93.16
$N_{cr}$ = 9.00

Figure 8: Pressure peak of NACA 4 series profile appears at point of maximum chamber (20% for NACA 4215)

In general a cubical spline between $[x_i, x_{i+1}]$ can be defined as:

$$P_i(x) = a_i + b_i(x - x_i) + c_i(x - x_i)^2 + d_i(x - x_i)^3 \tag{1}$$

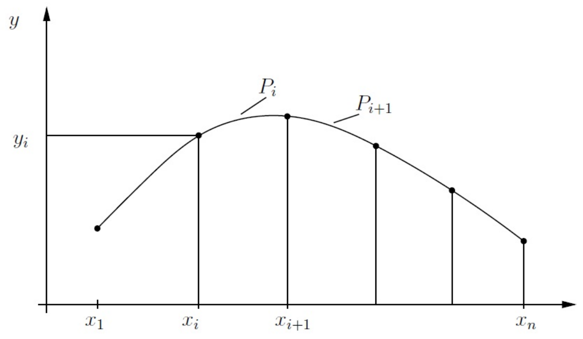


Figure 9: Cubical spline interpolation (Messnarz, 2011, p.57)

The task is to calculate the 4 coefficients $(a_i, b_i, c_i, d_i)$ that define the cubical polynomials. Therefore it is necessary to set up some boundary conditions. The facts there are 4 coefficients per polynomial and $(n-1)$ polynomials deliver $4(n-1)$ unknown coefficients.

According to Messnarz (2011, p.57) one can set up the following boundary conditions for each polynomial:

- The spline has to start at the first coordinate:

$$P_i(x_i) = y_i \tag{2}$$

- and has to end at the second coordinate:

$$P_i(x_{i+1}) = y_{i+1} \tag{3}$$

- At $x_{i+1}$, the first derivative of spline $i$ must be the same for spline $i+1$:

$$\frac{d}{dx}P_i(x_{i+1}) = \frac{d}{dx}P_{i+1}(x_{i+1}) \tag{4}$$

- and likewise is the second derivative:

$$\frac{d^2}{dx^2}P_i(x_{i+1}) = \frac{d^2}{dx^2}P_{i+1}(x_{i+1}) \tag{5}$$

These boundary conditions generate polynomials whose curvature is continuous. Even at a certain point $x_i$ where two splines intersect a continuous curvature is ensured. Due to the fact that for the first and last point one cannot define the boundary conditions for continuous first and second derivative two constraints are still missing. Therefore one has to manually introduce these remaining constraints. If the second derivatives at the very first and very last coordinate are set zero one obtains so called *natural cubical splines*:

$$\frac{d^2}{dx^2}P_1(x_1) = \frac{d^2}{dx^2}P_{n-1}(x_n) = 0 \tag{6}$$

(Messnarz, 2011, p.57)

With these boundary conditions it is possible to create a linear, inhomogeneous and tri-diagonal system of equations. Tri-diagonal means that only the main diagonal and the first diagonal above and below are non-zero. Solving this system of equations delivers all $4(n-1)$ coefficients. For more detailed information one is referred to look up detailed mathematical literature. For this purpose it should be enough to understand that cubical splines deliver a smooth surface and reduce the order of polynomials. Undulations and discontinuous surfaces would not be acceptable for aerodynamic investigation because they would falsify geometry and calculation of flow around the airfoil.

### 2.1.2 Deformation of Geometry

Now that the representation of geometry is set up it has to be defined what kind of deformation is suitable for the purpose of airfoil optimization. The easiest method is to dislocate the spline coordinates along the y-direction for fixed x-coordinates. Figure 10 depicts the dislocation of one spline coordinate:

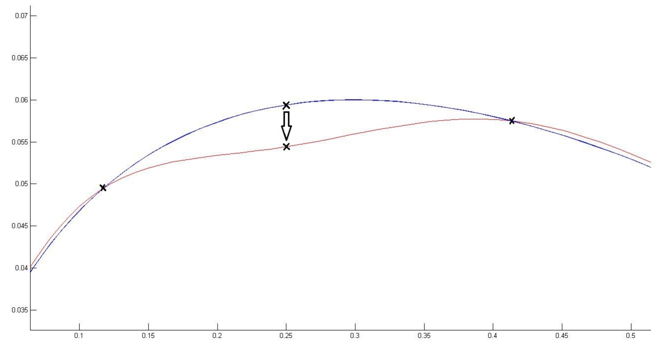


Figure 10: Deformation of airfoil by dislocating y-component of one spline coordinate

This simple method was also used for this project because first of all it is really easy because only y-components of coordinates vary but still full deformation of geometry can be conducted. Of course the generated profile also has to maintain a certain mechanical strength and should be technically producible. Therefore it is necessary to introduce certain deformation-constraints. It is also important to keep in mind here that XFOIL - the program that is used for the calculation of lift and drag (see Section 2.3) - requires a certain thickness at the trailing edge of the profile (depicted in Figure 11). Mark Drela, the author of XFOIL recommends a trailing edge thickness of $\sim 0.0005 * chord$. Therefore the two trailing edge coordinates are fixed coordinates, so dislocation for these coordinates is disabled. But before the mentioned constraints are introduced one has to think about the optimization method itself.

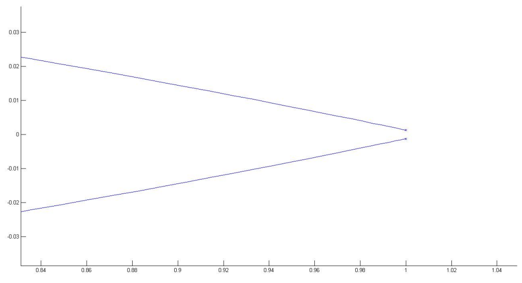


Figure 11: Trailing edge section of NACA 0012 airfoil as required by XFOIL

## 2.2 The Optimization Method

### 2.2.1 Introduction

The optimization method forms the core of the whole program. Several different methods are described in literature. For choosing the appropriate method it is necessary to define the problem. In mathematical terms the problem can be described as a **non-linear, constrained optimization problem** with **multiple design variables**. Non-linear means that the objective function (e.g. drag coefficient) is non-linear dependent on design variables (e.g. geometry). Constrained means that design variables must lie within certain boundaries (see Section 2.2.6) and finally due to the fact that there are several spline coordinates (already discussed in Section 2.1) that can be displaced there are multiple design variables (=spline coordinates).

Basically there are two different kinds of optimization methods distinguished by their general working principle:

- Gradient based algorithms
- Genetic algorithms

Both methods seem suitable for this kind of optimization problems and both methods have been used by different authors for the purpose of profile optimization. Each method has its advantages and disadvantages. Zingg et al. (2008) have done great work on comparing these two ideas on aerodynamical optimization.

Gradient based algorithms always incorporate, as one can already derive from the name, calculations of gradient. This means for each design variable of the design vector $\vec{X}$ one has to calculate the direction of displacement which leads to an improvement of the objective function. Once the gradient $\vec{g}$ is calculated there are several different opportunities to carry on. There are, for example:

- Conjugate gradient method
- Steepest descent method
- Quasi-Newton method

Each method then modifies the gradient differently with the goal to achieve rapid convergence to the optimum of the objective function, and returns a direction of descent $\vec{d}$ for each design variable. Finally an efficient stepwidth $\lambda$ has to be calculated which determines the width of dislocation (described in Section 2.2.5). At iteration $i$ one can update the design vector $\vec{X}$ according to:

$$\vec{X}_{i+1} = \vec{X}_i + \lambda_i \, \vec{d}_i \tag{7}$$

Genetic algorithms use the principle of evolution theory. The basic idea behind this is the following: The first step is to set up a population and to evaluate this population. Then pairs are formed (parents) whose properties are getting crossed over in new "children". Then comes "survival of fittest" and finally the loop closes with re-evaluation and finding new pairs until the objective function satisfies the convergence criteria. For in depth information one is referred to: “Holst et al. (2001) or Giannakoglou (2002)” (references adapted from Zingg et al. (2008, p. 105))

Further Zingg et al. state the following advantages and disadvantages of these two methods (2008, p. 106):

- Gradient based method
  - + Rapid convergence
  - + Gradient as a clear convergence criterion
  - + Computational costs approx. proportional to number of design variables
  - − Rather complex algorithm
  - − Intolerant for inaccurate gradients
  - − Often only local (not global) convergence
- Genetic method
  - + Global convergence
  - + Simple algorithm
  - + High tolerance regarding noisy objective functions
  - − Slow convergence
  - − Difficult to determine convergence criteria

The conclusion of their report is that depending on the purpose of use one can choose one method but they state that for specific and frequent use they would recommend gradient based methods due to lower computational costs compared to genetic algorithms. On the other hand, development-costs for gradient based algorithms usually are much higher.

Due to these facts but also because of personal favour a gradient based quasi-Newton method was implemented for this project which will be described closer in the next section.

If optimization of the drag coefficient with respect to a certain minimum required lift coefficient $C_L^*$ is performed, the problem can be stated as follows:

$$\underset{\vec{X},\alpha}{min}\, C_D(\vec{X},\alpha,M) \tag{8}$$

subject to (s.t.)

$$C_L(\vec{X},\alpha,M) \geq C_L^* \tag{9}$$

With $C_D$ the drag coefficient, $\vec{X}$ the design vector containing the spline coordinates, $\alpha$ the angle of attack, $M$ the Mach number, and $C_L$ defining the lift coefficient.

Hence the lift coefficient would already be a constraint for this optimization by stating that the lift coefficient always has to reach a certain prescribed minimum lift coefficient $C_L^*$.

The algorithm should now find the most feasible set of coordinates $\vec{X}$ that fulfil these requirements. Therefore the flow diagram in Figure 12 shows the necessary steps for the algorithm:

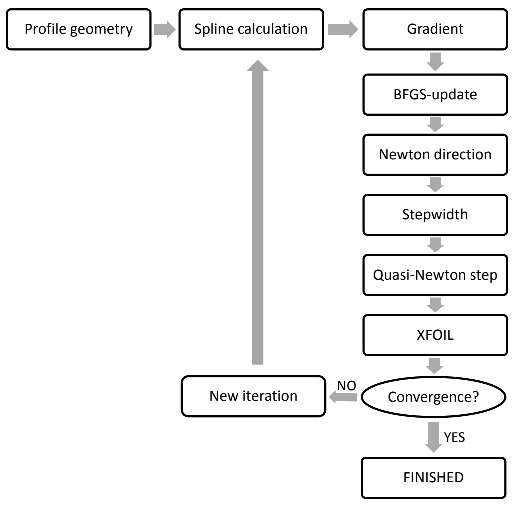


Figure 12: Flow diagram for quasi-Newton methods

This figure can also be seen as a guideline throughout this chapter. The implemented methods according to the flow diagram will be explained within the next sections.

### 2.2.2 Gradient

The calculation of the gradient $\vec{g}$ needs to be executed at least once within every iteration. The gradient is obtained by calculating the first derivative of the objective function, whereas angle of attack $\alpha$ and Mach number $M$ remain constant:

$$\vec{g} = \vec{\nabla} C_D(\vec{X}, \alpha, M) \tag{10}$$

With the design vector $\vec{X}$ containing x- and y-spline coordinates beginning and ending at the two trailing edge coordinates (actually $\vec{X}$ should be called design matrix but due to the fact that only the y-components are of interest one can also say that it is a vector):

$$\vec{X} = \begin{pmatrix} x_1 & y_1 \\ x_2 & y_2 \\ \vdots & \vdots \\ x_n & y_n \end{pmatrix}$$

Further, only the y-component of the spline coordinates shall be shifted. The gradient calculation reduces to differentiation with respect to the y-coordinates. To gather this gradient the method of finite differencing is implemented. There are different variations of this method as there are:

- Forward differencing
- Backward differencing
- Central differencing

The idea of finite differencing is to variate the independent variable around the original location for a small value $\epsilon$, calculate the change of the function and derive the gradient from that. As an example, equations (11) and (12) correspond to the forward and backward differencing scheme for spline coordinate $i$ after Taylor approximation:

$$f'_{fd} = \frac{C_D(y_i + \varepsilon) - C_D(y_i)}{\varepsilon} + \mathcal{O}(\varepsilon) \tag{11}$$

$$f'_{bd} = \frac{C_D(y_i) - C_D(y_i - \varepsilon)}{\varepsilon} + \mathcal{O}(\varepsilon) \tag{12}$$

Various different schemes have been tested throughout this project but the method of central differencing turned out to be most suitable. For coordinate $i$ the gradient with respect to y-spline-coordinates can be approximated similar to (11), again after Taylor approximation:

$$f'_{cd} = \frac{C_D(y_i + \varepsilon) - C_D(y_i - \varepsilon)}{2\varepsilon} + \mathcal{O}(\varepsilon^2) \tag{13}$$

Finally Figure 13 shows the differences graphically.

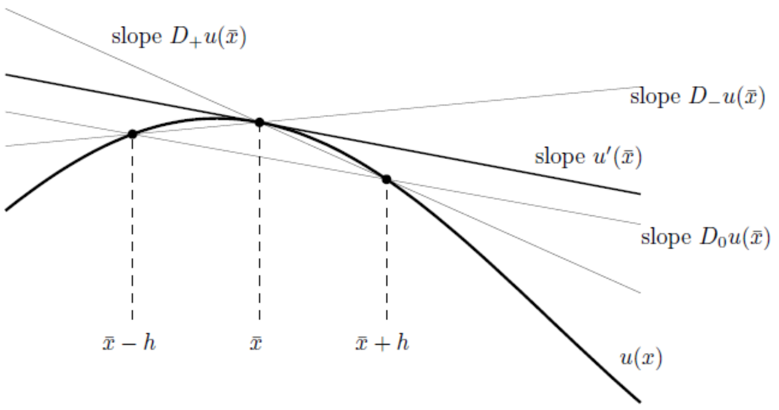


Figure 13: Forward differencing $D_+$ and backward differencing $D_-$ vs. central differencing method $D_0$ (LeVeque, 2006, p. 4)

According to the previously shown flow diagram (see Figure 12) the gradient is used for the calculation of the direction of descent. This step is explained within the next section.

### 2.2.3 Quasi-Newton Method

This section explains the heart of most optimization algorithms: The further processing of the gradient which was already calculated in the previous step. Although the gradient is already a descending direction, some methods promise a more efficient descending direction by applying better approximation models for objective functions. Simple methods only use the gradient as descending direction but more sophisticated methods also use the curvature for modelling the objective function. Mathematically the gradient is nothing else than a linear approximation of a function, whereas the Hessian and the gradient together form a quadratic approximation.

One example of such a more sophisticated, higher order method is the quasi-Newton method. This method was also chosen for the purpose of this project. It ensures rapid convergence of an objective function to an (often even global) optimum. Whether the algorithm converges globally or just locally should not be as important for this purpose of use due to rather strict constraints in terms of deformation of the geometry (see Section 2.2.6). For more detailed convergence analysis see Dennis' and Schnabel's (1996) work on optimization methods.

Before stepping on to the quasi-Newton method one has to take a step backward and think about single-variable optimization. The key to success is to approximate a function by Taylor approximation and solve optimization for the approximated model function $m$. Single variable optimization methods are widely known such as the Newton-Raphson method and similar ones. Figure 14 shows the idea the of Newton-Raphson method for a single variable root problem:

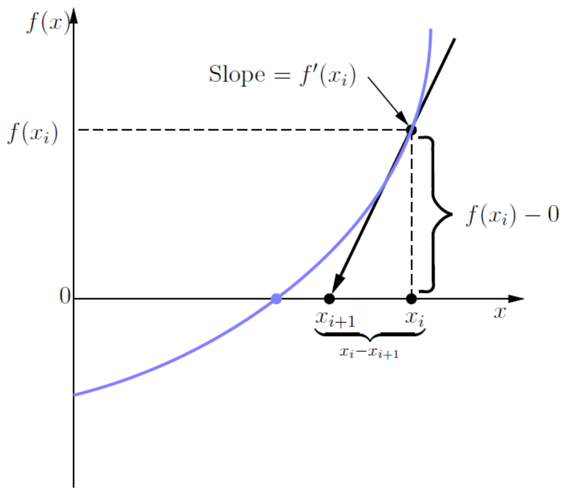


Figure 14: One iteration of Newton-Raphson method for single variable roots problem calculus (Messnarz, 2011, p. 30)

According to Dennis and Schnabel (1996, p. 100) any function can be approximated with Taylor polynomials by quitting after terms of second order to:

$$m(x+p) = f(x) + \nabla f(x)^T p + \frac{1}{2} p^T \nabla^2 f(x) p \tag{14}$$

(Dennis and Schnabel 1996, p. 100)

Now one can easily optimize the model by applying linear functional analysis. Differentiation and setting zero of (14) yields the minima of the model:

$$\frac{\partial m}{\partial p} = \nabla f(x) + p^T \nabla^2 f(x) \tag{15}$$

$$\nabla f(x) + p^T \nabla^2 f(x) = 0 \tag{16}$$

By reordering and replacing $p$ by $d$ in (16) one obtains the frequently used notation for the quasi-Newton method with direction of descent $d$ (called quasi-Newton direction). For iteration $k$ one has to solve a linear equations system (17) for $d^k$:

$$H_k d^k = -\nabla f(x^k), \tag{17}$$

where $H_k = \nabla^2 f(x^k)$ is the Hessian matrix of $f$, which contains all second partial derivatives, and $\nabla f(x^k) = \vec{g}_k$.

This information would already be sufficient in order to implement this method. But reconsidering (17) brings one to the point that in order to calculate the Hessian matrix of the objective function an unacceptable amount of function evaluations has to be performed during each iteration just for calculating the Hessian matrix.

$$H = \begin{pmatrix} \frac{\partial^2 f}{\partial x_1^2} & \cdots & \frac{\partial^2 f}{\partial x_1 \partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial x_n \partial x_1} & \cdots & \frac{\partial^2 f}{\partial x_n \partial x_n} \end{pmatrix} \tag{18}$$

If function evaluations can be performed rather quickly this may not seem such a big problem. But in case there were many independent variables or if function evaluation takes a lot of time, the efficiency of this method would not be suitable any more. For aerodynamic optimization function evaluations mean recalculation of aerodynamic properties every time. Now one can imagine that one single iteration will become very time consuming even if previous converged results were used for each calculation.

Additionally, calculating the second derivatives requires more than one function evaluation. Numerically one could approximate:

$$\frac{\partial^2 f}{\partial x_1 \partial x_2} \approx \frac{f(x_1+\varepsilon, x_2+\varepsilon) + f(x_1-\varepsilon, x_2-\varepsilon) - f(x_1+\varepsilon, x_2-\varepsilon) - f(x_1-\varepsilon, x_2+\varepsilon)}{4\varepsilon^2} \tag{19}$$

According to (19), 4 function evaluations are necessary for approximating second derivative of $f$. This makes an impressive amount of $16n^2$ function evaluations or $16n^2$ flow calculations for calculating the full Hessian matrix with $n$ being the number of spline coordinates. Therefore it seems feasible to not calculate the exact Hessian $H$ but better to approximate it.

### 2.2.4 Approximation of Hessian Matrix - BFGS Update

Within this section the BFGS-update (proposed by Broyden, Fletcher, Goldfarb and Shanno) is introduced as a method for Hessian approximation. The idea is to gather an approximation of $H$ with knowledge of change in gradient and change in variables. Therefore let

$$s = x^{k+1} - x^k \tag{20}$$

be the change in variables, and let

$$y = \nabla f(x^{k+1}) - \nabla f(x^k) \tag{21}$$

be the change of gradient from iteration $k$ to $k+1$. Now the BFGS-update can be performed according to

$$B = H + \frac{yy^T}{y^T s} - \frac{(Hs)(Hs)^T}{s^T H s} \tag{22}$$

(Dennis and Schnabel, 1996, p.212)

With $B$ defined as an approximated Hessian matrix. To ensure the direction of descent is still a descending direction a requirement for the quasi-Newton-method is a positive definite Hessian matrix. The BFGS-update ensures that positive definiteness is maintained as long as:

$$y^T s > 0 \tag{23}$$

(Dennis and Schnabel, 1996, p.102)

Interestingly it turned out to be most efficient for this purpose to initialize the Hessian matrix with unity matrix prior to the very first iteration. So it never is necessary to calculate the exact-Hessian matrix. Initializing Hessian matrix with unity matrix means that for the first iteration the negative gradient is equal to the direction of descent. After the first iteration the BFGS update can then be applied. If equation (23) cannot be fulfilled Dennis and Schnabel (1996) propose to try to modify the Hessian matrix $H$ or the approximated Hessian $B$ by:

$$H = H + \mu I \tag{24}$$

(Dennis and Schnabel, 1996, p.102)

With $\mu > 0$ a constant that gives more weight to the main diagonal and therefore ensures safe positive definiteness of $H$ which comes equivalent to positive curvature for functional analysis. In case even bigger values for $\mu$ cannot ensure $H$ or $B$ being positive definite the matrix has to be reinitialized with unity matrix $I$.

If one uses the exact Hessian matrix for calculation of direction of descent then the method is called *exact-Newton method*. It can be shown that under certain conditions both the exact- and quasi-Newton method show superlinear convergence (see Dennis and Schnabel (1996, chapter 6) for details). For superlinear convergence it is also important to calculate an efficient stepwidth. The methods that were used for calculation of stepwidth will be explained in the following section.

### 2.2.5 Stepwidth Evaluation

According to the flow diagram in Figure 12 the next step after the direction of descent is calculated is to find an efficient stepwidth $\lambda$ so that iteration $k$ can be finished with:

$$x^{k+1} = x^k + \lambda d^k \tag{25}$$

Finding an appropriate stepwidth is a critical point in optimization. The task is to find a stepwidth that provides sufficient decrease of the objective function but the stepsize must not be too big because this might also lead to slow convergence. Figure 15 depicts what might happen if stepsizes are too big. Dennis and Schnabel (1996, p.112) call equation (25) "Globally Convergent Modification of Newton's Method". For $\lambda = 1$, (25) is called Newton step. Further they state that the Newton-step ($\lambda = 1$) shall be executed whenever possible.

Basically there are different ways of obtaining efficient stepwidths. The most common ones are:

- Trust region
- Line search

The way line search methods work is like this report has assumed so far; after the gradient and a direction of descent has been calculated the stepwidth can be calculated which ensures maximum decrease of the objective function for a descending direction.

Trust region methods work upside down. According to an expected decrease of the objective function a stepwidth is assumed and afterwards the best suitable descending direction can be calculated. For more information on this method the

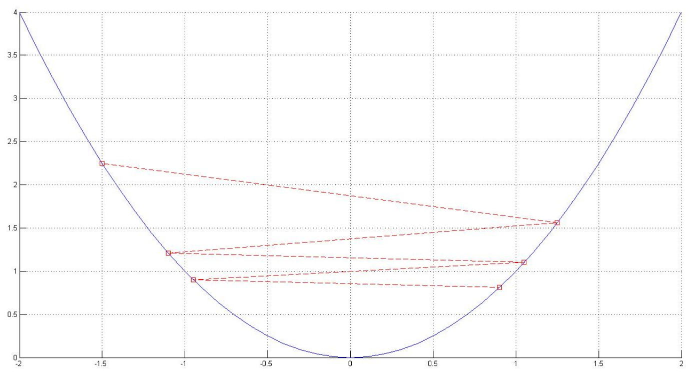


Figure 15: Too big stepwidths often cannot ensure sufficient decrease of objective functions

reader is referred to the literature. Further a Wolfe-Powell line search method will be discussed as it was implemented for this project.

According to Dennis and Schnabel (1996, pp.116-120) the two Wolfe-Powell conditions that ensure big and small enough stepwidths can be defined as follows. The first condition faces the problem of too big stepwidths which was depicted earlier in Figure 15:

$$f(x^k + \lambda^k d^k) \leq f(x^k) + c_1 \lambda^k \vec{\nabla} f(x^k) d^k \tag{26}$$

(Dennis and Schnabel, 1996, p.118)

Here $c_1 \in [0, 1]$ is a user defined constant that weighs this constraint. To describe this constraint in words it is noted that $\lambda^k \nabla f(x^k) d^k$ is the amount of descent of $f$ predicted by the local slope at $x^k$. This means that the actual descent in $f$ must not exceed a certain weighed, predicted descent.

Secondly one can easily imagine that the objective function will not decrease sufficiently by taking overly small steps. At this point condition number (27) gets active by stating:

$$\vec{\nabla} f(x^k + \lambda^k d^k) d^k \geq c_2 \vec{\nabla} f(x^k) d^k \tag{27}$$

(Dennis and Schnabel, 1996, p.118)

Hence $c_2$ is another user defined weight factor but now: $c_2 \in ]c_1, 1]$. This second constraint implies that the slope of $f$ at $x^k + \lambda^k d^k$ increases at least for a certain amount within one iteration.

The task is to find a stepwidth that fulfils both conditions. Therefore an algorithm from Geiger and Kanzow (1999, pp.45-47) was implemented and is presented now:

For simplification the first Wolfe-Powell condition (26) is reordered:

$$f(x^k + \lambda^k d^k) - f(x^k) - c_1 \lambda^k \vec{\nabla} f(x^k) d^k \leq 0 \tag{28}$$

With $t = \lambda^k d^k$ and $\psi = f(x^k + t) - f(x^k) - c_1 \vec{\nabla} f(x^k) t$ (28) is simplified to:

$$\psi(t) \leq 0 \tag{29}$$

Further with $\varphi'(t) = \vec{\nabla} f(x^k + \lambda^k d^k) d^k$ respectively $\varphi'(0) = \vec{\nabla} f(x^k) d^k$ the second Wolfe-Powell condition (27) simplifies to:

$$\varphi'(t) \geq c_2 \varphi'(0) \tag{30}$$

The following algorithm is adapted from Geiger and Kanzow (1999, pp.46-47). The algorithm can be split into two phases A and B:

## Wolfe-Powell Stepwidth Algorithm

Phase A:

(A.0) Choose $t_0 > 0$, $\gamma > 1$ and set $i := 0$

(A.1) If $\psi(t_i) \geq 0$ set $a := 0$, $b := t_i$ and continue with (B.0)
If $\psi(t_i) < 0$ and $\varphi'(t) \geq c_2 \varphi'(0)$, set *step* $:= t_i$ and cancel: **STOP 1**
If $\psi(t_i) < 0$ and $\varphi'(t) < c_2 \varphi'(0)$, set $t_{i+1} = \gamma t$, $i := i + 1$ and continue with (A.1)

Phase B:

(B.0) Set $j := 0$ and take $a_0 := a$ and $b_0 := b$ from PHASE A

(B.1) Set $t_j = \frac{1}{2}(a_j + b_j)$

(B.2) If $\psi(t_j) > 0$, set $a_{j+1} = a_j$, $b_{j+1} = t_j$, $j := j + 1$ and continue with (B.1)
If $\psi(t_j) < 0$ and $\varphi'(t_j) \geq c_2 \varphi'(0)$, set *step* $= t_j$ and cancel: **STOP 2**
IF $\psi(t_j) < 0$ and $\varphi'(t_j) < c_2 \varphi'(0)$, set $a_{j+1} := t_j$, $b_{j+1} := b_j$, $j := j + 1$ and continue with (B.1)

This algorithm returns an efficient stepwidth *step* at **STOP 1** or **STOP 2**. In some cases this algorithm ended up in infinite loops, producing infinitely small stepwidths. Therefore a loop cancelling point had to be included. The implementation can be found in file 'get_stepwidth_7.m'.

Having finished the description of the most important aspects of optimization methods, this leads to another important topic of optimization - constraints - as they will be discussed within the next section.

### 2.2.6 Constraints - The Penalty Method

Constraints is an important topic to allow user to gather a certain control over optimization processes. For aerodynamic optimization it is important to define constraints which ensure that a certain thickness and structural strength of airfoils is maintained. So for example one could define maximum dislocation in y-direction for each spline coordinate to always maintain a certain minimum thickness. Figure 16 depicts constrained degrees of freedom for some spline coordinates:

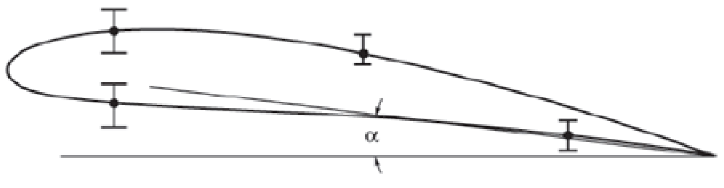


Figure 16: Some spline coordinates with constrained degree of freedom in y-direction (Huyse, 2001, p.3)

One can distinguish between *unconstrained optimization* as it was supposed until now and *constrained optimization* which is introduced now. Mathematically one would define the difference as:

- Unconstrained

$$\min_x f(x) \tag{31}$$

- Constrained

$$\min_x f(x) \tag{32}$$

$$\text{subject to: (e.g.) } x \geq x^* \tag{33}$$

Hence (33) constraints the optimization in this simple example. For demonstration let: $f(x) = x^4$ and $x^* = 2$. The task is now to introduce (33) into the optimization methods. As always there are different ways of achieving this. A really demonstrative method is the *penalty method.* This method was used for this project and will be discussed within this section.

Compared to other methods the penalty method has one big advantage; it allows to convert constrained optimization into unconstrained optimization. This is really helpful because all necessary tools for unconstrained optimization have already been discussed by now.

The clue is - as the name says - to introduce penalty functions that penalize objective functions if constraints are exceeded. In other words a (mainly local) minima is created in the area of constraints. Due to the fact that only the objective function is manipulated all other optimization methods do not have to be changed drastically.

How does the penalty method work mathematically? For implementing this method a so called penalty function $\phi$ is introduced. The function is defined by:

$$\phi(t) = \begin{cases} 0 & \text{for } t < 0 \\ kt^n & \text{for } t \geq 0 \end{cases} \tag{34}$$

In equation (34) $k$ and $n$ are positive, user defined constants that allow the user to control the strength of the penalty function. For the simple example $f(x) = x^4$ which was depicted earlier, constrained optimization with constraint $x \geq 2$ changes the objective function to (35) by adding the penalty function:

$$f^* = x^4 + \phi(2 - x) \tag{35}$$

For all $x < 2$ the penalty function gets active because $t = 2-x$ is smaller than zero in this case. The following figure depicts what the penalty means to the objective function. The modified objective function is depicted in red and here the penalty function was defined as $\phi(t) = 100t^3$.

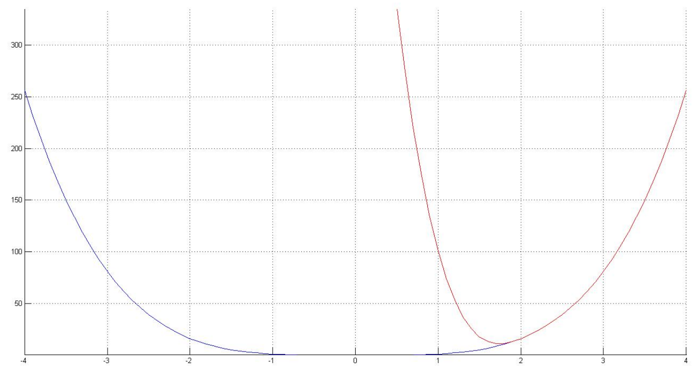


Figure 17: The penalty function gets active for all $x < 2$ and modifies the objective function such that a new optimum appears at $x \approx 2$. (here: $k = 100, n = 3$)

This simple one-dimensional optimization example shows very well how penalty methods work. In a similar way they work for higher dimensional functions. Of course then for each dimension a penalty exists which need to be summed up to gather a total penalty.

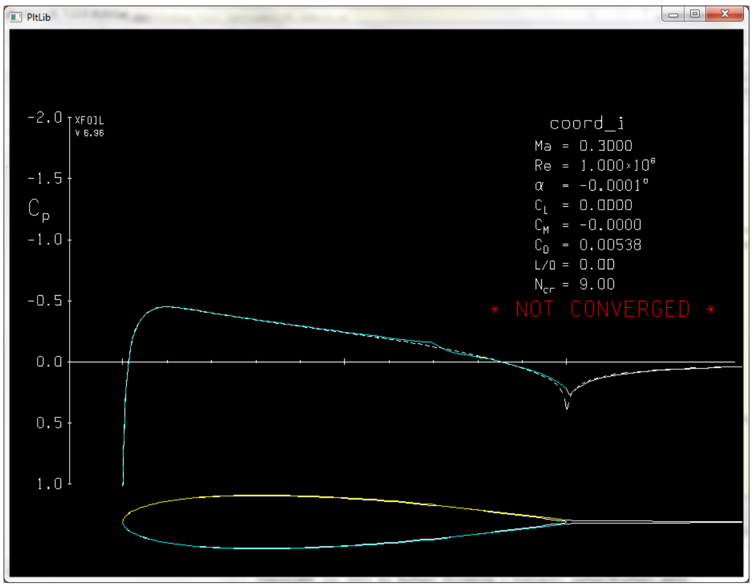
## 2.3 XFOIL

XFOIL is the program used for the flow calculations. The program is open source and the first version was developed by Mark Drela back in 1986. For this project version 6.96 published in 2000 was used. The latest version is 6.97 published in 2008 and is being developed any further. XFOIL is a well verified, widely known and used program with good reputation all over the world.

In general XFOIL is a panel code used for two-dimensional lift calculation in combination with boundary layer modelling for drag approximation around airfoils. The potential equations as a simplification of the full Navier-Stokes equations are solved. Potential equations neglect viscosity and compressibility. Therefore in order to calculate drag the boundary layer has to be approximated. According to the report of Drela and Giles (1986) this approximation is done by modelling the displacement thickness of boundary layers. Together these methods form a system of coupled non-linear equations which are solved by a global Newton method. For detailed background information on XFOIL methods see Drela and Giles (1986).

For the present purpose it is enough to keep in mind that compressibility effects (too large Mach numbers) and large areas of separation (too large angle of attacks) may spoil the results.

The method for drag calculation used by Drela also brought some difficulties. Often XFOIL could not find a solution for the non-linear equations system and XFOIL brought up the window shown in Figure 18. Re-calculations or more iterations do not help fixing this problem. Therefore an own routine was implemented which restarts the calculation with a slightly increased or decreased angle of attack ($\alpha \pm 0.001\,\text{deg}$) if XFOIL does not converge. In most cases this work-around helps very well.



```
coord_i
Ma  = 0.3000
Re  = 1.000×10^6
α   = -0.0001°
C_L = 0.0000
C_M = -0.0000
C_D = 0.00538
L/D = 0.00
N_cr = 9.00
* NOT CONVERGED *
```

Figure 18: XFOIL error message: XFOIL fails to converge drag calculation

After loading airfoil coordinates (either with internal NACA-airfoil generator or external .txt files) one can slightly modify or adjust geometry. It is possible to smooth the surface and to add flap deflection. It is noted again here that XFOIL requires a finite trailing edge thickness of approximately $0.0005 * chord$ (suggested by Mark Drela).

Finally, one can change to simulation mode with command (only type command after c>):

XFOIL c > oper

There are basically two different simulation modes; an inviscid and a viscous mode designated by (*.operi* respectively *.operv*). One can switch between these modes by typing *"visc"*. After specifying Reynolds- and Mach- numbers the simulation can be started. Here it is important to note that XFOIL defines Reynolds numbers and aerodynamic coefficients differently than usual! Usually these coefficients are dimensionless but in this case the dimensions always are $[Re, C_L, C_D, C_M] = m^{-1}$. Usually Reynolds numbers are defined as follows, whereas XFOIL defines:

$$Re = \frac{Vc}{\nu} \Rightarrow Re_{XFOIL} = \frac{V}{\nu} \tag{36}$$

Dimensionless factors are set in relation to chord length $c$. One can calculate "real" Reynolds numbers by multiplying $Re$ respectively other aerodynamic coefficients with the profile chord length $c$.

XFOIL allows starting a simulation either by prescribing a certain AOA (or sequence of AOA's) or by prescribing a certain lift coefficient $C_L^*$. By doing so XFOIL automatically determines the necessary AOA for this specific $C_L^*$. This makes it very simple to incorporate lift constraints by always starting the simulation with the command ($C_L = 0.5$; only type command after c>):

.OPERv c > *cl* 0.5

Note that XFOIL commands are case insensitive.

### 2.3.1 XFOIL-MATLAB Interface

Flow calculations are executed in XFOIL whereas the optimization process is controlled from MATLAB. Therefore XFOIL simulations have to be started from MATLAB. A XFOIL-MATLAB interface which allows running XFOIL from MATLAB and accessing data from XFOIL-simulations was already implemented by Rafael Oliveira in 2011. This interface (XFOIL - MATLAB interface v1.0) is freely available on the MATLAB file exchange server and was used for this project after slight modification.

The interface itself consists of the function 'def_XFOIL' saved in file 'runXFOIL.m' and the files in folders '@Airfoil' and '@XFOIL'. All modifications can be controlled in function 'def_XFOIL'. For example it is possible to toggle XFOIL simulation window on and off by setting the variable 'xf.Visible = true/false'. Further internal filtering for getting smoother surfaces can be activated under 'xf.addFiltering'. XFOIL solves the equations system for boundary layers iteratively and therefore one can set a maximum number of iterations via the variable 'xf.addIter'. By setting too few iterations, XFOIL will often not converge. Any number between $100 - 200$ iterations may be appropriate. Finally it is possible to plot the polars from XFOIL in MATLAB if the corresponding lines in file 'runXFOIL.m' are uncommented.

This is the end of the methods section. At this point all methods have been introduced that are necessary for the implemented optimization method. At the beginning the geometric representation with cubical splines was discussed before the optimization method was introduced. To execute the quasi-Newton method a second order approximation of the objective function was made. With approximation of second partial derivatives (Hessian matrix) via BFGS-update only the gradient have to be calculated with the central differencing method. As a last point a Wolfe-Powell stepwidth algorithm was introduced before the unconstrained optimization was changed to constrained optimization by implementing a penalty method.

In the following results section two examples for optimization will be described where all necessary handles and settings of these methods are explained step by step.

# 3 Results

Within this section two different optimizations will be executed for demonstration. The necessary settings will be explained step by step. Going through these examples will give one the necessary handles to execute own optimization with this program. The first example deals with a simple case where no difficulties are expected:

## 3.1 Example 1

Consider a simple example where a NACA 0012 profile forms the basis for optimization with 11 spline coordinates. With 11 spline coordinates the method should have no difficulties with finding an optimum because the number of optimization parameters is not too high. For more spline coordinates it may happen that the method converges only slowly or even not at all (see example 2).

### Creating spline coordinates

The first step is to provide spline coordinates that are used as control points for optimization. Therefore one can manually save x- and y-components of spline coordinates as column vectors in a text file. The second possibility is to extract equally spaced coordinates from an existing profile geometry. Therefore it is necessary to provide coordinates of a profile also in a .txt file. With the script called 'get_spline_coordinates.m' one can extract a certain number of spline coordinates by doing following:

1 Save coordinate-text-file to MATLAB working directory (finite trailing edge thickness reqired!)

2 Open 'get_spline_coordinates.m' and set the number of spline coordinates in variable *num* to 11

3 Help MATLAB finding the coordinate-text-file by giving the name of the file in *NACA_coord = load('filename.txt')*

4 Executing the script will save spline coordinates in a text-file called 'spl_coord.txt'

5 Check geometry of profile and location of splines visually in the popup-figure (e.g. Figure 19)

After performing these steps an appropriate number of spline coordinates has been extracted and the following popup-figure appears for visual control:

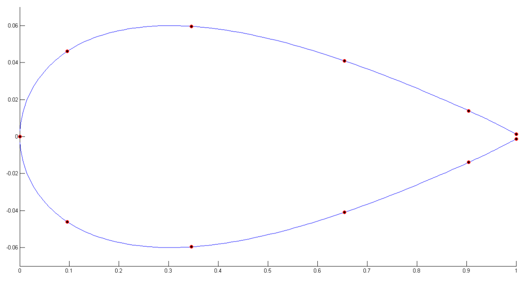


Figure 19: Popup-figure for visual control of spline coordinates.

**Hint:** In case the result shown in popup-figure 19 is not satisfying one has to ensure that enough profile coordinates were provided as input in file 'get_spline_coordinates.m'.

## Setting up and running the optimization

Now one needs the spline-coordinate-text-file as input for optimization. All important settings can be made in the file 'start_opti.m'. Step by step one has to make the following settings:

1 Copy the 'spl_coord.txt' file to the working directory of 'start_opti.m'

2 In file 'start_opti.m': Select the method for stepwidth calculation. Here the Wolfe-Powell-method is most suitable. Therefore set: '*method = 1*'

3 Help MATLAB finding the spline-coordinate text file by giving the name of the file in: '*NACA_coord = load('spl_coord.txt')*'

4 Set the lift coefficient to an arbitrary value. now set: '*CL = 0*'

5 Set the Reynolds number as defined by XFOIL related to the chord length (also see Subsection 2.3). Now set: '*Reynolds = 1e6*'

6 Define the Mach number by: '*Mach = 0.3*'

7 The next step is to define the number of coordinates '*num*' for interpolation. This is the number of coordinates that will be loaded for the XFOIL-simulation. This number should not be too low. Now set: '*num = 250*'

8 Now set the constraints for optimization: '*dislocation_in*' respectively '*dislocation_out*' define the boundaries for maximum outward and inward displacement of the spline coordinates in percent of the original y-components. Here set: '*dislocation_in* = 0.3' and '*dislocation_out* = 0.1'

9 The penalty function needs a reference for dislocation. Therefore one can manually define a reference geometry. This will become important for optimization with multiple runs as will be discussed in example 2. For now set: '$reference\_spline = user\_spline$'

10 Defining appropriate convergence criteria is very difficult for this method because one can hardly predict optimization. Despite some variables can be defined that give the user more control over the simulation. Set:

●*min_decrease* $= 0.25$: Expected relative decrease of drag coefficient (25% expected)

●*eps_grad* $= 1e-3$: Minimum required value for absolute sum of gradient (optimization is considered converged if the gradient falls below a small value 'eps_grad')

●*max_iter* $= 50$: The conditions for 'min_decrease' and 'eps_grad' must be fulfilled simultaneously whereas 'max_iter' cancels optimization after a certain iteration.

11 In file 'runXFOIL.m' under function 'def_XFOIL.m' one can toggle the visibility of the XFOIL simulation window on or off by setting the variable '*xf.Visible = true*'

12 Now one can start the optimization by running 'start_opti.m'. After the first iteration several figures will pop up for displaying the progress of the optimization.

While the optimization is running the control figures will update every iteration giving an user information about the progress of optimization. Note that MATLAB executions always can be quit by pressing 'CTRL+C' in the MATLAB console!
For this example it will take about 40 iterations until an optimum has been found. Figure 20 shows the popup figure for this optimization. One can see that $C_D$ beautifully converges to an optimum value found for $C_D = 0.00383$. The fact that the drag coefficient of the original value was $C_{D_0} = 0.00567$ brings an impressive improvement of 33% for $Mach = 0.3$, $Reynolds = 10^6$ and $C_L = 0$.

The coordinates for the optimized profile can be accessed via text files that are created during optimization. If the optimization is quit because the convergence criteria were fulfilled, the spline coordinates can be found in file 'opti_spl.txt' and the interpolated coordinates are saved to file 'opti_coord.txt'. If the optimization is quit because XFOIL did not converge or the user cancelled manually, the coordinates for the current iteration are stored to file 'coord_new.txt'. In this case the spline coordinates can only be accessed via the MATLAB workspace.

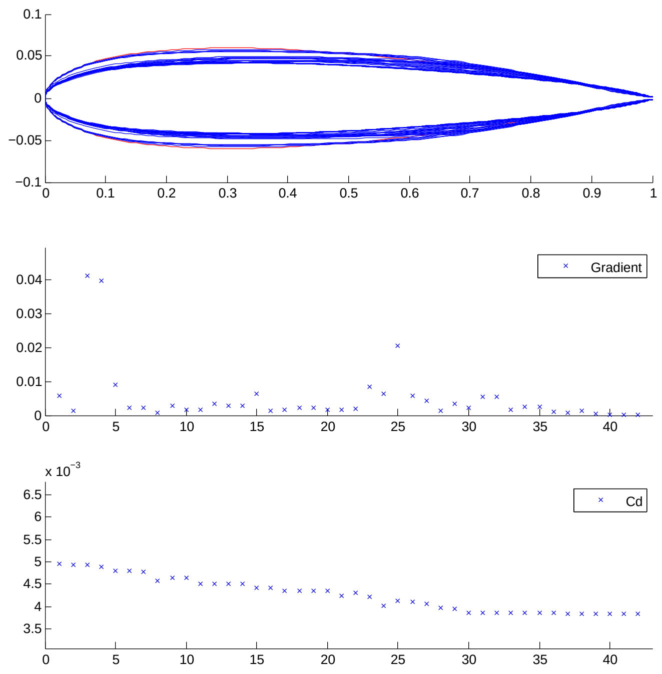


Figure 20: Popup window during simulation shows change in geometry, gradient and drag coefficient

The trend of the gradient depicted in the second subfigure in Figure 20 indicates that it is very difficult to determine whether the method has converged or not. Theory shows that optimization has converged if $\nabla f = 0$ but often the gradient gets close to zero before it rises again and further descent is possible. This is why no general conclusion on convergence criteria could be drawn. For different cases the user has to find out what works best.

Finally Figure 21 shows the detailed geometry after the last iteration. Depicted in red is the original NACA 0012 profile; blue is the optimized profile after 42 iterations and the dashed lines in black depict the outer and inner constraints (10% respectively 30%). The fact that these constraints were chosen rather loose is one reason for the huge improvement of the drag coefficient.

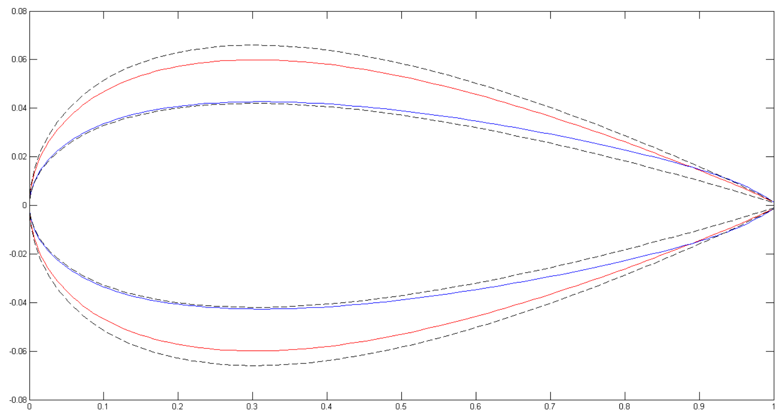


Figure 21: Detailed geometry of the optimized profile after 42 iterations. Depicted in red is the original NACA 0012 profile; blue is the optimized profile after 42 iterations and the dashed lines in black depict the outer and inner constraints (10% respectively 30%).

## 3.2 Example 2

This example will deal with the case when there are more spline coordinates. In example 1 the number of spline coordinates was 11. For this example this number is increased to 19 spline coordinates. Of course one could proceed exactly as in example 1 but in some cases the method will only converge very slowly or will not converge to better values than with 11 splines. To avoid this, one has to pre-converge with less spline coordinates (e.g. 11) and refine the number of spline coordinates step by step. This example will guide through this process step by step. For basic settings see example 1.

### Pre-convergence

Consider again a NACA 0012 profile as basis profile. Now let $C_L = 0.3$ and the number of spline coordinates for the first run shall be 11. To load these coordinates set, after creating the appropriate coordinate files:

- user_spl = load ('0012_11.txt')
- reference_spl = user_spl

The reference splines are needed for the penalty function as a reference for constraints. For the first run the user_spl coordinates shall be the reference as well but in the second run when the optimized profile is used as starting geometry still the original NACA 0012 profile shall be the reference for constraints. Therefore one can load extra geometry as reference which differs from 'user_spl'. This will become important for the refinements later on. The other settings are:

- Wolfe Powell line-search method
- $C_L = 0.3$
- Reynolds = 6.28e6
- Mach = 0.3
- num = 250
- dislocation_in = 0.2
- dislocation_out = 0.1
- min_decrease = 0.2
- eps_grad = 5e-3
- max_iter = 20

With those settings one can see that at iteration 6 XFOIL cannot converge any more because of uncontrolled, too big deformation. For this example this should not be a problem because Figure 22, which depicts further optimization, shows that the profile has already converged sufficiently so that the result already can be taken as basis for further runs.

***Note:*** *If one likes to continue optimization at this point but the stepwidth method produces geometries that cannot be solved by XFOIL, one should use the 'custom stepwidth method'. Therefore one manually has to change the value of the variable 'method' to 2 by typing 'method = 2' into the MATLAB command window. Now the algorithm uses the custom stepwidth method. Of course it is not necessary to start from the beginning again but evaluating the second cell in file 'start_opti.m' (where iteration i begins) allows continuing with the current iteration. But if the algorithm has not completed one single iteration at this point (if the popup window has not appeared yet) it is necessary to recompute the whole script (the algorithm has not reached the second cell, respectively 'iteration i', yet)!*

*If one wants to switch back to the other method the algorithm has to be cancelled manually by pressing 'CTRL + C' and the method has to be switched to 'method = 1' again. This is highly recommended because usually the Wolfe-Powell-method is more efficient than the custom stepwidth function.*

Continuing with custom stepwidth function brings the following result after convergence has been obtained after 14 iterations. The objective function decreased by 21%.

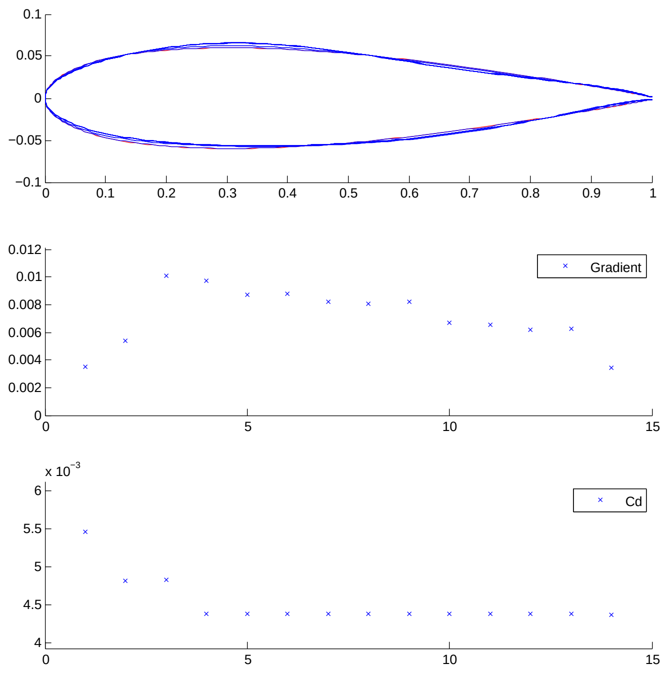


Figure 22: Result of pre-convergence with 11 spline coordinates; optimization is converged after 14 iterations

### Refinement 1

In case the algorithm has quit because XFOIL failed to converge the coordinates of the last iteration are stored in the file 'coord_new.txt'. Otherwise the coordinates are stored in the file 'opti_coord.txt'. Use this file now to extract the next higher number of spline coordinates with MATLAB file 'get_spline_coordinates.m' (see example 1 for detailed steps). Here the next step was chosen with 15 spline coordinates. Note that the penalty function needs reference splines for constraints and of course the penalties should still refer to the original NACA 0012 profile. To give these spline coordinates as an input one has to create another text-file containing the next higher number of spline coordinates from the original NACA 0012 coordinates. Here the new inputs are:

- user_spl = load ('fine1_15.txt')

- reference_spl = load ('0012_15.txt');

It may happen that XFOIL fails to converge right at the beginning. Therefore one has to start with custom stepwidth but it is recommended to switch back manually to Wolfe-Powell method after one or two iterations. For this run the input was set to:

- method = 2
- user_spl = load('fine1_15.txt')
- reference_spl = load('0012_15.txt')
- $C_L$ = 0.3
- Reynolds = 6.28e6
- Mach = 0.3
- num = 250
- dislocation_in = 0.2
- dislocation_out = 0.1
- min_decrease = 0.15
- eps_grad = 3e-3
- max_iter = 40

After 16 iterations this foil was further optimized for another 15% as Figure 23 shows.

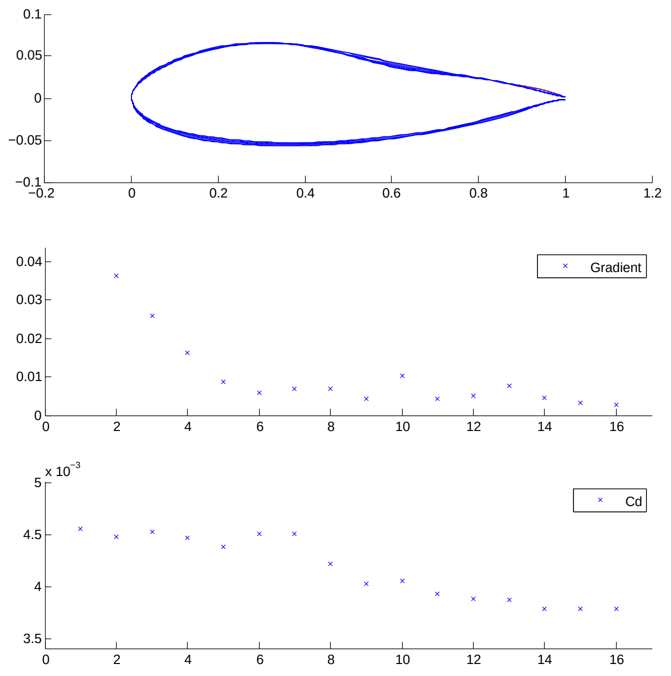


Figure 23: Result of refinement with 15 spline coordinates; optimization has converged after 16 iterations

### Refinement 2

Finally one last refinement with 19 spline coordinates is executed. Therefore again create a new spline coordinate text file from the optimized profile (either saved in coord_new.txt if manually quit or opti_coord.txt if the convergence criteria were fulfilled) and another reference spline coordinate text file (e.g. '0012_19.txt') from the original NACA 0012 profile. Both now with 19 spline coordinates. The input is similar to refinement 1 but now a lower gradient is requested to ensure good convergence. On the other hand not so much improvement is expected any more.

- method = 1
- user_spl = load('fine2_19.txt')
- reference_spl = load('0012_19.txt')

- dislocation_in = 0.2
- dislocation_out = 0.1
- min_decrease = 0.05
- eps_grad = 1e-3
- max_iter = 50

The final result (Figure 24) shows up after 29 iterations when the optimization was quit manually because of little change in gradient and drag coefficient. The drag coefficient was reduced to $C_D = 0.00368$. Due to the fact the original NACA 0012 profile drag coefficient is $C_{D_0} = 0.00556$ the profile drag was minimized for remarkable 33,8%.

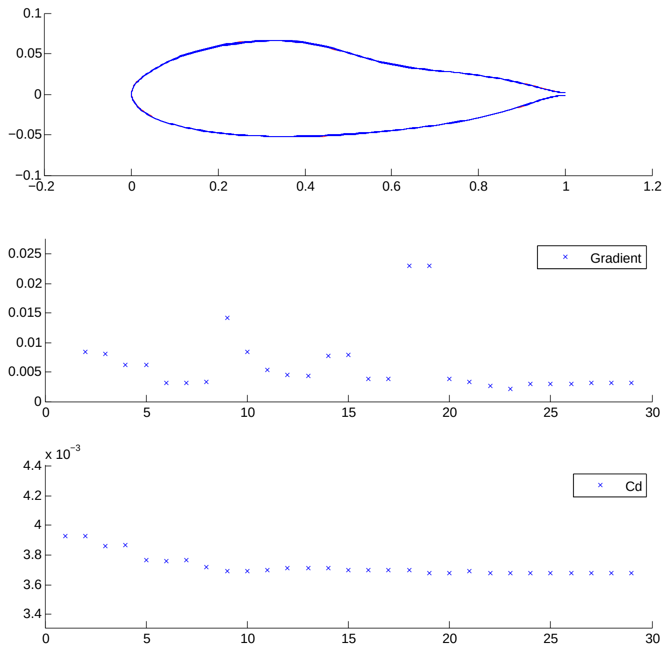


Figure 24: The last refinement with 19 spline coordinates is quit after 29 iterations, optimizing the foil for another 8%

Figure 25 shows the new geometry (blue) in detail compared to the NACA 0012 profile (red) and the boundaries (black):

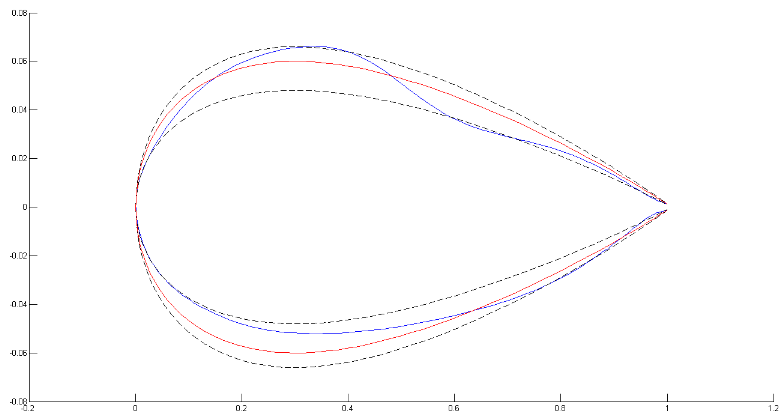


Figure 25: Detailed geometry of optimized NACA 0012 profile for $Re = 6.28 * 10^6$, $M = 0.3$, $C_L = 0.3$ and 19 spline coordinates

# 4 Discussion

The goal of this thesis was to implement different mathematical methods to allow simple aerodynamic optimization of arbitrary airfoils. Throughout this report the reader got introduced into all methods and the necessary handles were described as well. Within this section all these tools will be reflected from a critical point of view.

## 4.1 Discussion of Methods

At the beginning the method of cubical splines for geometric representation of airfoils was introduced. This method works well unless there are not too few spline coordinates because then geometry may not be representative any more. The problem is that the current version of the program only allows fixed trailing edge coordinates but no other fixed coordinates that can be defined by the user. On the other hand, if one had more fixed coordinates only very little displacements were possible because big displacements would cause undulating surfaces.

As a further step the methods for mathematical optimization - beginning with the gradient - got introduced: The central differencing method turned out to be best working on this but unluckily the results strongly depend on the stepsize of variation. In the end a stepsize of $\epsilon = 0.001$ was chosen. Further, another uncomely modification had to be performed due to stability reasons. Too often the optimization ended in strongly deformed airfoils which could not be solved by XFOIL. To avoid this the gradient was defused by a factor $\frac{1}{100}$. This is why one can find the gradient calculation implemented to:

$$g_i = \frac{C_D(y_i + \epsilon) - C_D(y_i - \epsilon)}{200\epsilon} \tag{37}$$

The quasi-Newton method was chosen for calculating the direction of descent. Therefore, besides the first partial derivatives, the second partial derivatives (Hessian matrix) of the objective function or at least an approximation of it were necessary. For optimization it was necessary that the Hessian matrix is safely positive definite to ensure the new optima being a minimum. The initialization was performed with the unity matrix.

A critical point is the method for stepwidth calculation. Remember the user defined constants $c_1$ and $c_2$ in equations (26) and (27) that gave the user control over the stepsize. In some cases this method produced too big stepwidths which led to too big deformations that could not be solved by XFOIL any more. Therefore a custom stepwidth function was implemented which can be switched on and off manually by the user.

As a last point the optimization was constrained by penalty terms. The user again can influence the strength of these terms by changing factor and exponent of the penalty function. The spline coordinates usually will not hurt the constraints but in some cases the interpolated splines might hurt the boundaries. To avoid this one would have to implement higher resolution constraints where not only the spline coordinates but all interpolated coordinates are constrained.

## 4.2 Discussion of Results

As a result two showcase examples were executed. The second one was a bit more complex and required several steps taken by the user. This example also showed the limits of this method - at a number of 19 spline coordinates only little or none improvement was possible.

One big drawback of airfoil optimization are its off-design characteristics. Usually aircraft get optimized for cruising conditions but under slightly different conditions optimization is non profitable any more and characteristics become even worse than for the unoptimized airfoil. Figure 26 shows off-design $C_L$ for the optimized NACA profile and Figure 27 shows off-design speed for an RAE 2822 profile.

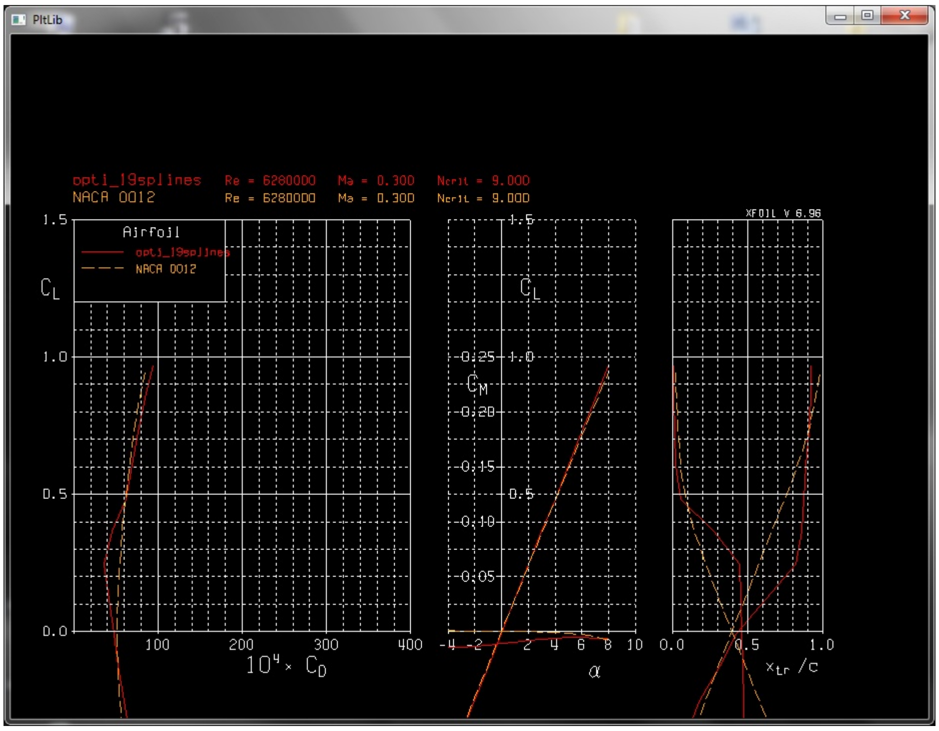


Figure 26: Polar of the optimized NACA 0012 profile with 19 spline coordinates

Optimized airfoils are very sensitive to changes in Mach numbers and lift coefficients. Multipoint optimization would be an opportunity to avoid bad off-design properties. Therefore one no longer executes optimization only for one single design point but for multiple design points. Finally one has to find a compromise between different optima. This allows to extend the area of optimization as Figure 27 shows:

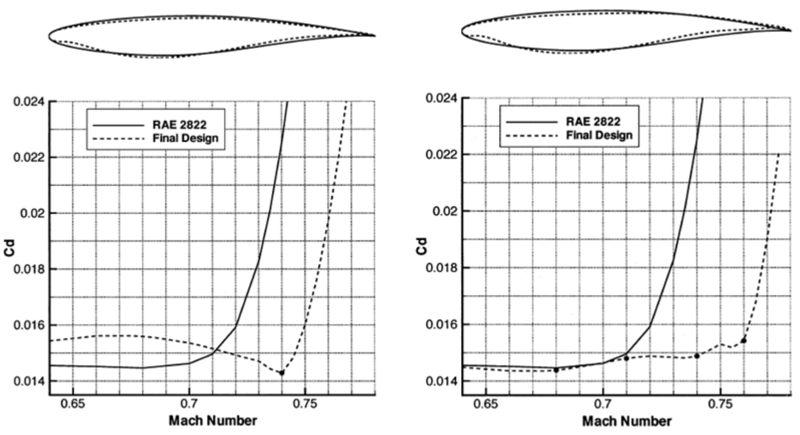


Figure 27: Multipoint optimization allows to extend the optimized area (Nemec and Zingg, 2004, p. 1063)

The right plot in Figure 26 is an interesting point as it shows the location of the transition point from the laminar to the turbulent boundary layer. As one can see, this point was shifted backwards for approximately 20-30% for the optimized profile in the area of the design lift coefficient. This means that the laminar boundary layer was extended as it is the case with laminar-profiles. The characteristics of the optimized profile come similar to laminar-profiles. Displacement of the transition point is one reason for the huge improvement of the drag coefficient.

The biggest problem which is not solved yet is to determine whether the optimization has already converged sufficiently or not. The problem is that the gradient does not converge as beautifully to zero as theory would promise. Theoretically optimization is finished when the gradient approaches zero but the high non-linearities of the objective function cause big undulations of the gradient as example 2 has shown before. Because of that it is strongly up to the user to find good convergence criteria or to quit optimization manually. But example 2 also has shown that once the gradient approaches small values only little improvement is possible. So it is up to the user if one wants to set strict convergence criteria and put high effort for little improvement or if one sets loose convergence criteria and is satisfied with not totally converged solutions.

# 5 Summary and Outlook

The results and the discussion have shown that the implemented optimization method works great on optimization for 10 to 17 spline coordinates but lacks efficiency for more than 19 coordinates but this may be in the nature of things. 19 or more degrees of freedom for sure are not easy to handle. But the author thinks that there is no necessity to add much more than 19 spline coordinates. Otherwise one would have to think of different approaches for geometrical representation or other optimization methods.

The goal, to implement an easy to use and rapidly working tool for optimization was fully achieved. This very first version of the method allows single point, constrained and unconstrained optimization of arbitrary profiles. Nevertheless there are some possible future improvements for further versions. It has already been mentioned that the convergence criteria is not working very reliable. Too often this criteria strongly depends on the actual case. This is certainly not the intention of convergence criteria and it would be necessary to find an opportunity to define convergence criteria that works independently from individual cases.

Further it is quite laborious to manually switch between different stepwidth strategies if one method fails. The user would certainly like the program to decide on its own which stepwidth strategy is most suitable. The custom stepwidth strategy was just implemented for continuing if the Wolfe-Powell method produces nonsensical geometry but it works not as efficient as the Wolfe-Powell method does.

Another possible improvement has already been mentioned in the previous Discussion: multipoint optimization allows to extend the design point to a design area. In reality the design point will hardly ever occur exactly as predicted. Therefore it is necessary to optimize an airfoil for several design points or a design area. This would definitely be a reasonable and important add-on for further versions of this method.

Other opportunities for optimization than discussed in this thesis would be optimization of lift coefficient $C_L$ or aerodynamic quality $\frac{C_L}{C_D}$. Exchanging these optimization goals would be very easy because these values get calculated by XFOIL anyway. By doing so one could maximize the achieved range or other important characteristics of airfoils.

If one wants to maximize the lift coefficient XFOIL will not be suitable any more because for large angles of attack and high lift coefficients large areas of separation will occur which will falsify XFOIL-results. Therefore it was necessary to incorporate for example ANSYS into the optimization method because only this kind of software can calculate lift and drag sufficiently for high angles of attack with large areas of separation. But this would cause new problems because one has to ensure that the mesh-quality is sufficient throughout the whole optimization process. Further the optimization process would last much longer because usually calculations with ANSYS will never be as fast as calculations with XFOIL. And with several hundreds of flow calculations per optimization this method will become very time consuming.

This very last section has shown that there are some possible improvements and add-ons for future research on this method but for the very first version of this method the author is fully satisfied with the results.

# References/Bibliography

Abbott, I., Doenhoff, A., and Stivers, L., "Summary of Airfoil Data", National Advisory Committee for Aeronautics, Report No. 824, 1945.

Anderson, J. D., *Fundamentals of Aerodynamics: Fifth Edition in SI Units*, McGraw-Hill Series in Aeronautical and Aerospace Engineering, New York, 2011.

Dennis, J.E., Schnabel, R., *Numerical Methods for Unconstrained Optimization and Nonlinear Equations,* Classics in Mathematics, Society for Industrial and Applied Mathematics, Philadelphia, 1996.

Drela, M., Giles, M.B., "Viscous-Inviscid Analysis of Transonic and Low Reynolds Number Airfoils", AAIA Journal, Vol. 25, No. 25, 1986.

Geiger, C., Kanzow, C., *Numerische Verfahren zur Lösung unrestringierter Optimierungsaufgaben*, Springer Verlag, Berlin, 1999.

Giannakoglou K., "Design of optimal aerodynamic shapes using stochastic optimization methods and computational intelligence", Progress in Aerospace Sciences, vol. 38, p. 43-76, 2002.

Holst T. L., Pulliam T. H., "Aerodynamic Shape Optimization Using A Real-Number-Encoded Genetic Algorithm", Technical Report No. 2001-2473, AIAA, June, 2001.

Huyse, L., "Free-form Airfoil Shape Optimization Under Uncertainty Using Maximum Expected Value and Second-order Second-moment Strategies", NASA/CR-2001-211020, ICASE Report No. 2001-18.

LeVeque, R., "Finite Difference Methods for Differential Equations", University of Washington, Washington, 2006.

Li, W., Huyse, L., and Padula, S., "Robust Airfoil Optimization to Achieve Consistent Drag Reduction Over a Mach Range", NASA/CR-2001-211042, ICASE Report No. 2001-22.

Messnarz, B., "Angewandte Mathematik 3: Skriptum zur Vorlesung", Fachhochschule Joanneum, Graz, 2011.

Nemec, M., Zingg, D.W., "Multipoint and Multi-Objective Aerodynamic Shape Optimization", AIAA Journal, Vol. 42, No. 6, June 2004.

Selig, M., and Maughmer, M., "Generalized Multipoint Inverse Airfoil Design", AIAA Journal, Vol. 30, No. 11, November 1992.

Zingg, D.W., Nemec, M., Pullioam, T.H., "A Comparative Evaluation of Genetic and Gradient-Based Algorithms Applied to Aerodynamic Optimization", University of Toronto, Toronto, 2008.