Design and Analysis of Low Reynolds Number Airfoils

Final Project
MATH 6514: Industrial Math

Submitted to Dr. J. McCuan
4 December 2002

By Nicholas K. Borer

1. Introduction

The most important part of heavier-than-air vehicles is the mechanism used to get them off the ground and to keep them there. This can come from brute force, i.e. propulsive power, such as vectored thrust or rocket engines, or can come from the more efficient means of lifting bodies. The latter approach is the most common nowadays, and the applications of some sort of lifting body has been used on the smallest MEMS system to the largest super-transport. By far the most common lifting body in effect today is the wing, a concept embodied in biology as well as modern machinery. Wings of all shapes and sizes lift bodies into the air, but special care must be made to ensure that these wings are of the proper geometry. If not, efficiency, stability, and performance will suffer.

In the past, the two-dimensional slices of a wing, dubbed airfoils, were designed via experimental means. Thousands of airfoils have been cataloged to date as seen in [1] and [2]. These airfoils are grouped into families based on flow regime, designer, and other factors. An engineer can then pick the best airfoil based on its experimentally determined characteristics for any given application.

The recent explosion in computational power has made possible the design of custom airfoils for most applications a realizable task without much experimental effort. This is through the backbone of physics, and most often found through first defining a desirable pressure distribution and matching this to an airfoil geometry. The question of how to determine the design pressure distribution becomes paramount in the design engineer's mind. Off-design performance is often a consideration as well, and this often results in conflicting objectives.

This project deals with the design, analysis, and application of an airfoil to fit a real-world engineering problem. The sections that follow describe the design problem and its criteria, a method to approach this design problem, validation of this method, and finally the results of the simulation.

2. Application Overview

The design problem selected for this project is the design of a low-Reynolds number (100,000 to 1,000,000) airfoil to be used on Uninhabited Aerial Vehicles or UAVs. The motivation for this study is the application of this airfoil or series of airfoils to a vehicle to be designed, built, and flown by members of Georgia Tech's Design / Build / Fly team. This team is participating in the American Institute for Aeronautics and Astronautics (AIAA) competition of the same name. This year's contest rules can be found in [3].

At this time, the author is undecided as to which specific configuration will be utilized for this study. However, some preliminary work has been done on a flying wing or

blended-wing-body (BWB) configuration. These initial studies indicate that a vehicle with a maximum takeoff gross weight (W_{TO}) of under 13 pounds, a reference wing area (S) of 4.0 square feet, and an aspect ratio of 4.0 is capable of best meeting the contest rules as outlined in [3]. Some of these preliminary analyses can be seen below in Figure 1 in the form of a design plot. The contours on this plot show required wing loading versus thrust-to-weight ratio for a 120-foot takeoff ground roll and several cruise speeds. Note that for this contest it is *necessary* to meet the takeoff requirement and *desirable* to have the maximum possible speed – itself a tradeoff, as these requirements indicate opposite trends in wing loading [4].

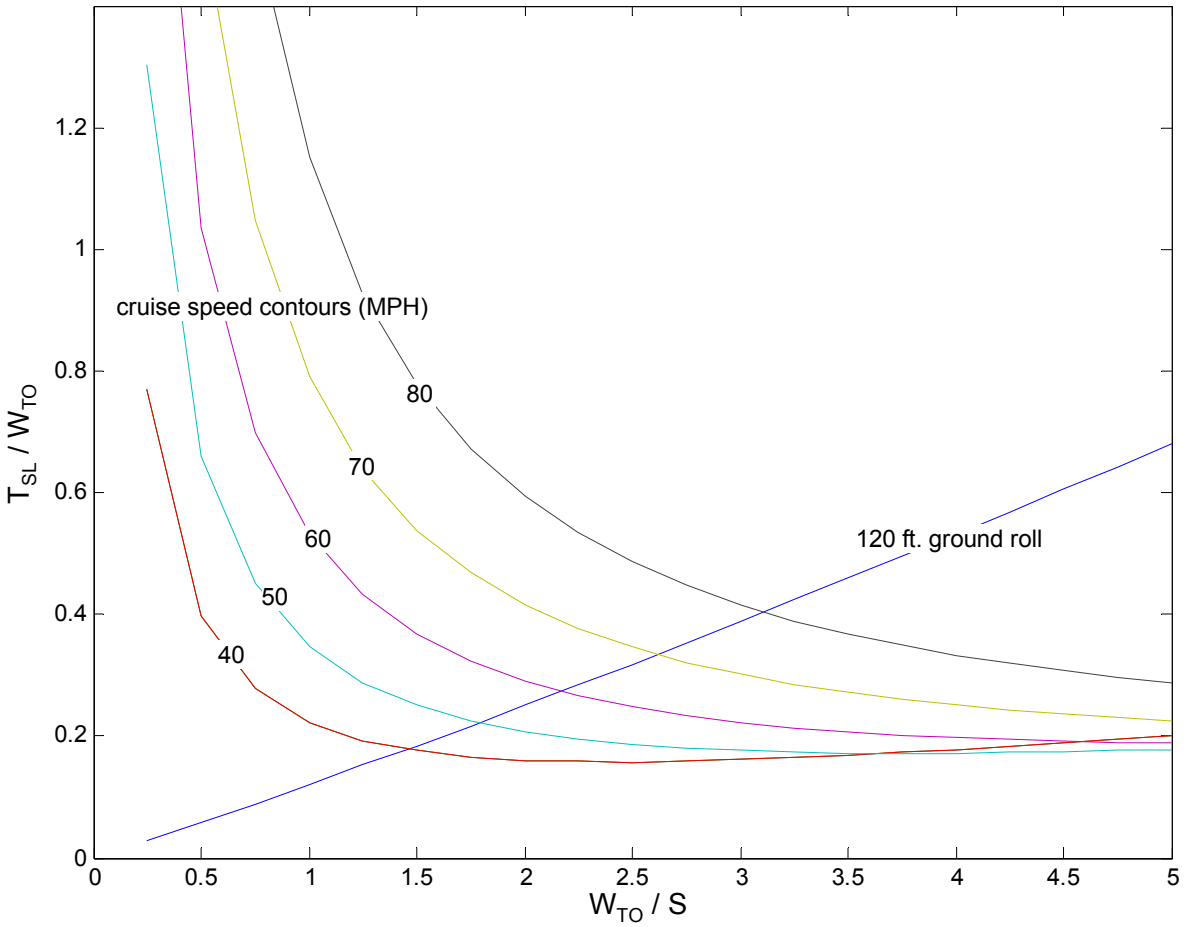

Figure 1. *Design Plot for a Flying Wing / BWB Configuration*

This problem is an exciting one because it involves many elements other than simply matching the best pressure distribution. Off-design conditions and inherent stability are key players in the proper design of the airfoil(s) for this aircraft. Eventually, this design may fly, providing a measure of validation for the results generated over the course of this project.

3. Design Conditions and Constraints

Preliminary studies indicate that this aircraft will takeoff at around 35 miles per hour (freestream) and have a max cruise speed of approximately 75 to 80 miles per hour. With the geometry mentioned above, this indicates that the airfoil should have satisfactory characteristics for Reynolds numbers from 200,000 to 700,000. This range can be generalized to extend from 100,000 to 1,000,000.

The preliminary design point for this vehicle is at a freestream velocity of 75 miles per hour at sea level standard (SLS) conditions. This indicates a wing design lift coefficient (C_L) of approximately 0.24 – quite a low number. This design condition will most likely not fall at the maximum lift-to-drag ratio of the vehicle, as is often the case with racing vehicles. The objective of the airfoil design will be to minimize the induced drag in this configuration with the proper camber, thickness, and twist distribution across the wing for a given planform. Some constraints may be pitching moment characteristics (stability) and trim drag limits.

One other important regime for this aircraft will be at takeoff and landing. These two conditions will take place at slightly above the stall speed of the aircraft, somewhere in the vicinity of 35 miles per hour (freestream). Design criteria here will be an acceptable maximum wing lift coefficient (C_{Lmax}), at this point estimated at 1.2, while maintaining adequate pitch and drag characteristics. The propulsive system for this vehicle has been tentatively identified and will thus be able to give some insight into the performance of the aircraft in this high-drag configuration. The specific excess power (P_s) is a possible metric as well, as it is a measure of the vehicle's acceleration and rate-of-climb capabilities [5].

One important consideration is that the Reynolds number range indicated includes the range typically listed at natural transition from laminar to turbulent flows. This has serious implications on the design of the airfoil, as now natural *and* pressure gradient-induced transition need to be quantified.

These are but a few of the several things to consider when setting up a design program and defining objective criteria. The objective of this project is not nearly as far-reaching, however – all that is trying to be done in this exercise is the creation, validation, and execution of an inviscid airfoil design routine.

4. Airfoil Analysis

The first step in airfoil design is choosing a method that has the proper balance of fidelity and speed for the given application. As such, a variety of computational analysis methods are available to the airfoil designer. These range from linear methods, concerned with solving the velocity potential equation, to more complicated methods that involve solving the Euler (inviscid) or Navier-Stokes (viscous) equations at various

points on and around the airfoil to determine the nature of the flow. The linear methods are much faster but much more limiting in application while the flow solver methods, often referred to as Computational Fluid Dynamics (CFD) are less limiting but more computationally intense to solve.

The airfoil to be designed over the course of this project is planned for application into a subsonic low-Reynolds number environment as described earlier. Therefore, the otherwise limited linear methods can be used to determine the airfoil characteristics. Specifically, a vortex panel method will be employed for the calculation of lift and pitching moment properties, while a boundary layer analysis will be necessary for some of the peculiarities associated with a low-Reynolds number flow.

4.1 Panel Methods

The vortex panel method belongs to a class of more general methods known as panel methods. These methods work by stating that the governing flow equations can be solved via a superposition of elementary flows. These elementary flows can be sources, sinks, doublets, vortices, and others. The elementary flows are placed at control points along the perimeter of a body, typically at the center of a straight segment. Lifting bodies can only be approximated with vortex flows, while nonlifting (symmetrical) flow can be approximated with sources and sinks, among others. The vortex allows for the creation of circulation, an abstraction quite necessary for lifting flowfields.

4.2 Vortex Panel Theory

The main assumptions of all panel methods are incompressible, irrotational flow, often called potential flow. These two assumptions greatly simplify the governing equations of fluid motion.

In two dimensions, the continuity equation is given by

$$\frac{\partial \rho}{\partial t} + u\frac{\partial \rho}{\partial x} + v\frac{\partial \rho}{\partial y} + \rho\left(\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y}\right) = 0 \tag{1}$$

where ρ is the fluid density, u is the velocity component in the x-direction, and v is the velocity component in the y-direction. For incompressible flow (constant density), the continuity equation simplifies to

$$\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} = 0 \tag{2}$$

Furthermore, irrotationality, which implies inviscid flow, is defined in 2-D as

$$\frac{\partial u}{\partial y} - \frac{\partial v}{\partial x} = 0 \tag{3}$$

One can define a velocity potential Φ such that

$$\begin{aligned} \frac{\partial \Phi}{\partial x} &= u \\ \frac{\partial \Phi}{\partial y} &= v \end{aligned} \tag{4}$$

These equations satisfy the condition of irrotationality. The velocity potential can be substituted into the continuity equation, yielding

$$\frac{\partial^2 \Phi}{\partial x^2} + \frac{\partial^2 \Phi}{\partial y^2} = 0 \tag{5}$$

This is a 2-D version of the Laplace equation. Similarly, one can define the stream function Ψ as

$$\begin{aligned} \frac{\partial \Psi}{\partial y} &= u \\ -\frac{\partial \Psi}{\partial x} &= v \end{aligned} \tag{6}$$

The stream function can also be substituted into the continuity equation to yield a 2-D form of the Laplace equation. The stream function is related to the velocity potential in that equipotential lines (lines of constant velocity potential) are perpendicular to streamlines (lines of constant stream function).

The principle of superposition states that the solution to the Laplace equation can be found through the superposition of elementary flows along an equipotential line or streamline. Therefore, several elementary flows positioned along a streamline can be used to solve a problem in potential flow. The vortex panel method utilizes the elementary solution to vortex flow and uniform flow distributed along a streamline. The stream function for vortex flow is given by

$$\Psi = \frac{\Gamma}{2\pi} \ln(r) \tag{7}$$

where Γ is the strength of the vortex (a constant that is positive clockwise), and r is the distance between the position of the vortex and the point where the velocity is evaluated. The velocity components for a vortex, in cylindrical coordinates, are

$$v_r = \frac{1}{r}\frac{\partial \Psi}{\partial \theta} = 0$$
$$v_\theta = -\frac{\partial \Psi}{\partial r} = -\frac{\Gamma}{2\pi r} \tag{8}$$

where v_r is the radial velocity component and v_θ is the tangential velocity component. If the vortex is placed at a location r_0 (where r_0 represents a point (x_0,y_0) in 2-D Cartesian space), the stream function becomes

$$\Psi = \frac{\Gamma}{2\pi}\ln\left(\left|r - r_0\right|\right) \tag{9}$$

where

$$\left|r - r_0\right| = \sqrt{\left(x - x_0\right)^2 + \left(y - y_0\right)^2} \tag{10}$$

As the vortex solution is singular, it should be made as weak as possible to produce a smooth flowfield composed of many elementary solutions. Therefore, the vortex panel method uses vortices of infinitesimal strength $\gamma_0 ds_0$, where ds_0 is the length of a small segment of the airfoil, and γ_0 is the vortex strength per unit length. Then, the stream function due to all such infinitesimal vortices at a point in space may be written as

$$\oint \frac{\gamma_0}{2\pi}\ln\left(\left|r - r_0\right|\right)ds_0 \tag{11}$$

where the integral is evaluated over all the vortex elements on the airfoil surface.

The stream function for uniform flow is much simpler and is given generally by

$$\Psi = V_\infty y \tag{12}$$

where V_∞ is the freestream velocity. The addition of all of these effects yields

$$\Psi = u_\infty y - v_\infty x - \frac{1}{2\pi}\oint \gamma_0 \ln\left(\left|r - r_0\right|\right)ds_0 \tag{13}$$

which is the stream function at any point in 2-D space.

Panel methods are evaluated only at the streamline on the surface of the airfoil as the body surface is a streamline. Therefore, the stream function value will be some constant value C, and equation (13) becomes

$$\Psi = u_\infty y - v_\infty x - \frac{1}{2\pi}\oint \gamma_0 \ln\left(\left|r - r_0\right|\right) ds_0 - C = 0 \tag{14}$$

over all points on the surface of the airfoil.

At this point, it is possible to give a physical interpretation to γ_0. The vortex panel method uses this infinitesimal vortex strength to model a tangential velocity over the surface of the airfoil. This is directly analogous to the pressure over a surface in inviscid, incompressible flow via Bernoulli's equation, given by

$$p + \frac{1}{2}\rho\left(u^2 + v^2\right) = p_\infty + \frac{1}{2}\rho_\infty V_\infty^2 \tag{15}$$

where p is the static pressure at a point on the surface of the airfoil. Furthermore, p_∞ and ρ_∞ are the static pressure and fluid density of the freestream. The static pressure is often nondimensionalized by the freestream dynamic pressure, which yields

$$C_p = \frac{p - p_\infty}{\frac{1}{2}\rho_\infty V_\infty^2} = 1 - \frac{u^2 + v^2}{V_\infty} = 1 - \left(\frac{V^2}{V_\infty^2}\right) \tag{16}$$

where C_p is known as the pressure coefficient. This is a measure of the difference in static pressure over the surface of the airfoil from freestream conditions, which is important because this can be used to determine the pressure forces (lift) on an airfoil in incompressible flow. As discussed above, the infinitesimal vortex strength γ_0 is directly related to the velocity at a point on the surface of the airfoil, so this can be substituted into equation (16) to yield

$$C_p = 1 - \frac{\gamma_0^2}{V_\infty^2} \tag{17}$$

One more simplification is necessary before the numerical solution to equation (14) can be found. The pressure at the trailing edge of the airfoil must be equal and the velocity must leave smoothly in the same direction at the upper and lower edge. This is stated as

$$\gamma_{0_{TE-upper}} = -\gamma_{0_{TE-lower}} \tag{18}$$

where the subscripts *TE-upper* and *TE-lower* denote points on the upper and lower surface of the trailing edge, respectively. This is known as the Kutta condition.

Now, equation (14) can be solved numerically. First, the body is divided into n segments, called panels. The flow velocity changes rapidly near the leading and trailing

edges, so it is a good idea to increase the panel density near these regions. Next, control points are created at the center of each segment. These control points indicate where each vortex will be placed. The strength of each of these vortices is assumed constant over each panel and are taken as unknowns. This yields

$$u_\infty y_i - v_\infty x_i - \sum_{j=1}^{n} \frac{\gamma_{0,j}}{2\pi} \int_j \ln\left(\left|r - r_0\right|\right) ds_0 - C = 0 \tag{19}$$

where the index i refers to the control point where equation (14) is applied and j refers to the panel over which the line integral is evaluated. The integrals over the individual panels depend only on the panel shape, which, as a straight line segment, is determined by the panel end points. This indicates that the integral may be computed analytically. The resulting quantity is defined as follows.

$$A_{i,j} = \frac{1}{2\pi} \int \ln\left(\left|r - r_0\right|\right) ds_0 \tag{20}$$

Here, $A_{i,j}$ is known as the influence of panel j on index i. Equation (19) becomes

$$u_\infty y_i - v_\infty x_i - \sum_{j=1}^{n} A_{i,j} \gamma_{0,j} - C = 0 \tag{21}$$

This equation may be written at each of the n control points (end points). However, there are n+1 total unknowns: the n infinitesimal vortex strengths $\gamma_{0,j}$ and the constant C. Therefore, one more equation is necessary. This is the Kutta condition, written as

$$\gamma_{0,1} + \gamma_{0,n} = 0 \tag{22}$$

Here, the panels are read in from the trailing edge around the lower surface to the leading edge, then back over the upper surface to the trailing edge. Finally, one is left with a system of n+1 equations and n+1 unknowns that can be easily inverted and solved.

This leaves the user to a variety of post-processing options. The pressure coefficients can be found from equation (17) at each control point, and from here the pressure distribution can be analyzed and processed to determine the lift and pitching moment. Note that in 2-D, incompressible, inviscid flow there will be no drag. This is why vortex panel codes are often combined with boundary layer analyses to determine the viscous drag of the airfoil sections. References [6] and [7] contain a far more detailed explanation of the vortex panel method.

The specific analysis code used for this assignment was a panel code written in Matlab made available to the class at the beginning of the semester. It can be found on the

course web site [8]. The final results were validated with XFOIL, a two-dimensional vortex panel code that is also capable of a simple boundary layer analysis [9]. The theory and capabilities of XFOIL are described in greater detail in [10].

5. Validation of Panel Code

It is always a good idea to test the results of a theory against known data. Therefore, several test cases were performed with empirical data obtained from a trusted source. This was compared with the output from the Matlab-based panel code written for this design exercise as well as for XFOIL.

5.1 Test Case

The test case prescribed was to determine the lift and pitching moment characteristics of a NACA 0015 airfoil and to compare the results to published data. The data was generated for this airfoil at a Mach number of 0.29 and a Reynolds number of 1.95 million. This approximates a subsonic condition, suitable for a fully inviscid analysis. The data was found from reference [8].

The experimental data was compared against three different analyses. First, the Matlab panel code was run with a Prandtl-Glauert compressibility correction added in. This was compared to the XFOIL case with its built-in Karman-Tsien compressibility correction. Finally, the experimental data was compared to XFOIL's viscous corrections via its internal boundary layer model. Each case was evaluated at angles of attack from one to eleven degrees in increments of a half degree.

5.2 Test Results

The results of the validation cases were visualized in several ways. Figure 2 displays the pressure distribution found for the Matlab and XFOIL inviscid cases as well as the XFOIL viscous case. Unfortunately, there was no data on hand to compare this pressure distribution to the actual pressure distribution used to generate the empirical data used for the other tests. As expected, all three solutions are rather comparable. The XFOIL viscous solution does show evidence of boundary layer transition or perhaps a laminar separation bubble at an x/c of approximately 0.12 on the upper surface. This, of course, cannot be picked up by the inviscid Matlab or XFOIL result.

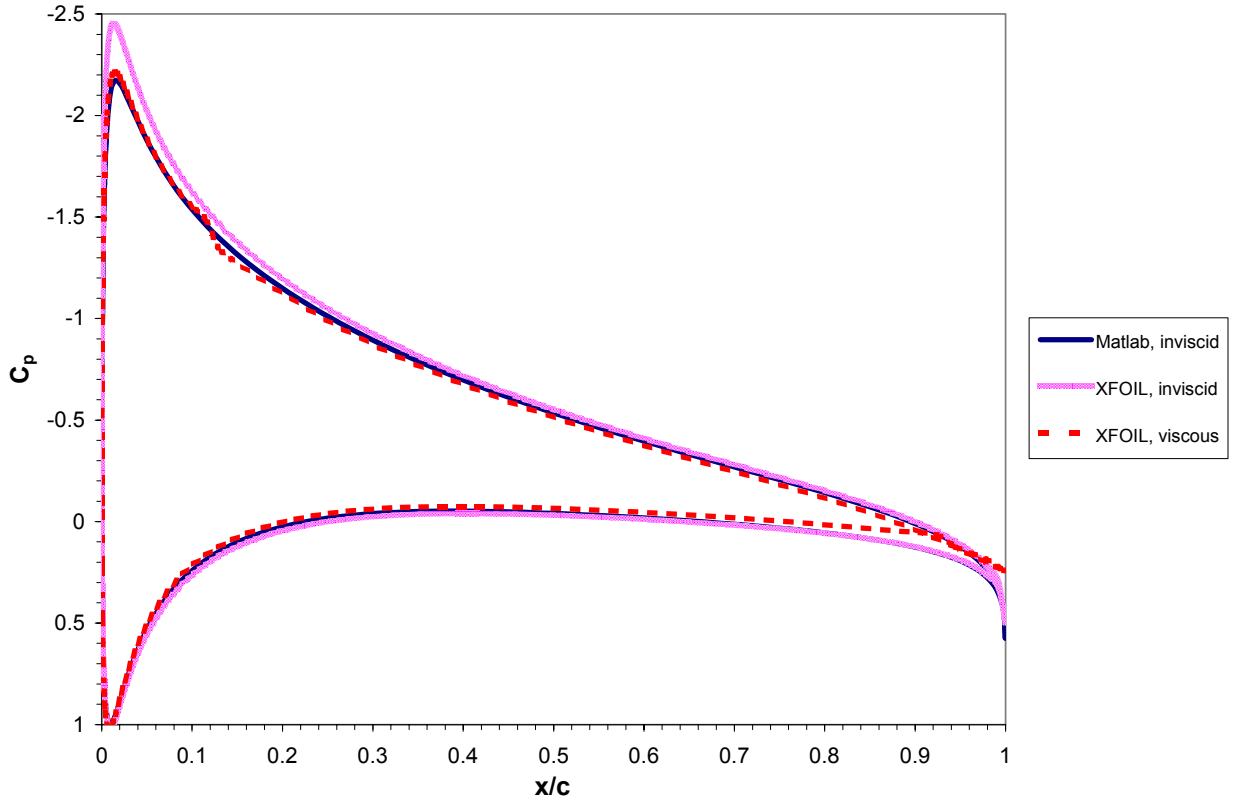

Figure 2. *Calculated Inviscid and Viscous Pressure Distributions for a NACA 0015 Airfoil*

The comparison of the lift characteristics between the test cases with the experimental data yielded some interesting results. All three validation cases were reasonably close to the experimental data at small angles of attack, but all overpredicted lift as angle of attack increased. The worst was the XFOIL inviscid case with the compressibility correction, while the inviscid Matlab case was only marginally better. This is probably due to the difference between the Karmen-Tsien compressibility correction used by XFOIL and the more conservative Prandtl-Glauert correction used in the Matlab code. The viscous case was better still, but still overpredicted lift at high angles of attack. Figure 3 shows the comparison of the lift characteristics to the experimental data.

Overprediction of lift is to be expected from this type of analysis. This is because the panel method, even when combined with a boundary layer analysis, assumes that the flow is 100% attached. A thicker airfoil such as this will exhibit separation at the trailing edge as the angle of attack is increased, which will tend to decrease the lift and increase the drag. Analysis of the boundary layer can be used to predict where separation will occur, but the only way to accurately predict the loss of lift and drag produced by trailing edge separation is through solution of the Navier-Stokes equations throughout the flowfield. This is a computationally expensive process and will thus be avoided for this project.

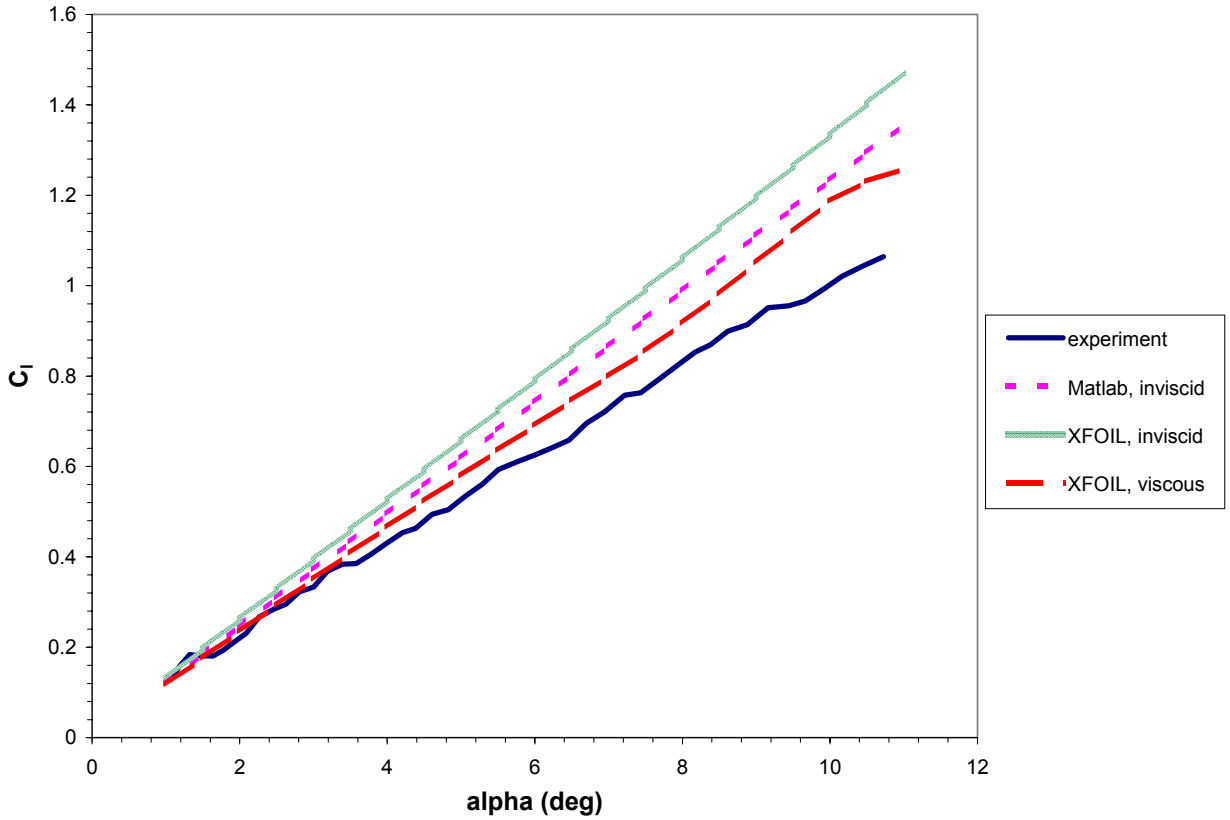

Figure 3. *Comparison of Test Cases to Experimental Data: Section Lift Coefficient versus Angle of Attack*

The section moment coefficient data had some problems as well. The comparison of the analyses to the experimental data is shown in Figure 4. As seen in the figure, the experimental data itself is quite noisy and thus its accuracy is suspect. All cases exhibit the same trends toward a more nose-down pitching moment at higher angles of attack, a phenomena that indicates that the quarter chord, where this moment measurement was taken, is not the aerodynamic center of the airfoil. In practice, the moment coefficient should remain relatively constant at the aerodynamic center until stall, when the slope will become sharply negative.

Another interesting note is that the experimental data and test data indicate that the moment coefficient is a non-zero value throughout this range of angle of attack. This is further evidence that the quarter chord of this airfoil is not an accurate measure of the aerodynamic center, because in theory a symmetric airfoil such as this should have a pitching moment coefficient of zero at this location.

Though not perfect, these results indicated that the inviscid Matlab and XFOIL viscous solutions would work well for the small angles of attack (within five degrees) encountered during design for cruise conditions

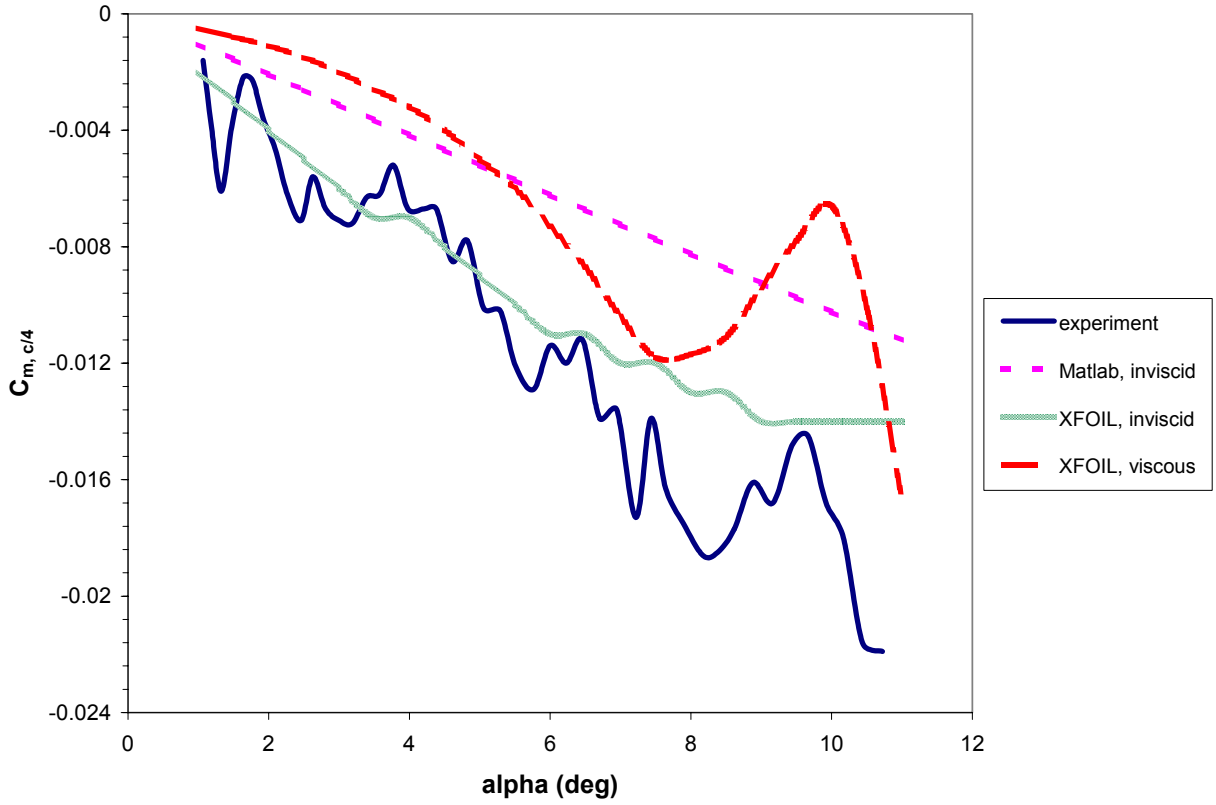

Figure 4. *Comparison of Test Cases to Experimental Data: Section Moment Coefficient versus Angle of Attack*

6. Airfoil Design Methods

In the past, airfoils were designed experimentally. A designer would build a model based on experience, place it in a wind tunnel, record the results, and make minor adjustments to the airfoil model if it did not perform satisfactorily. Often, these airfoils were classified into several families that could be placed into a catalogue of airfoils. The aircraft designer would then choose the appropriate airfoil or airfoils for an aircraft based on the recorded experimental data.

The advent of the digital computer has changed this process significantly. Now, an engineer can make custom airfoils to fit almost any design condition and wing planform within the realm of physics. There are three approaches used for custom airfoil design: trial and error, optimization, and inverse design. The first is rather self-explanatory and was the method of designing custom airfoils in the past. The optimization technique is akin to an automated trial-and-error scheme, where the objective is to match an airfoil geometry to an input pressure distribution. The final design scheme, inverse design, involves specifying an input pressure distribution and creating the geometry from there. Of these, the optimization method was selected due to the author's experience with optimization techniques.

6.1 Optimization Method Overview

The optimization method employed for this project involves five principal steps. These are (1) define target pressure distribution and seed airfoil; (2) pre-process inputs and initialize design variables; (3) optimize design variables and terminate when optimization criteria are met; and (4) post-process data and determine fitness of solution. The flow of information for this process is shown in Figure 5.

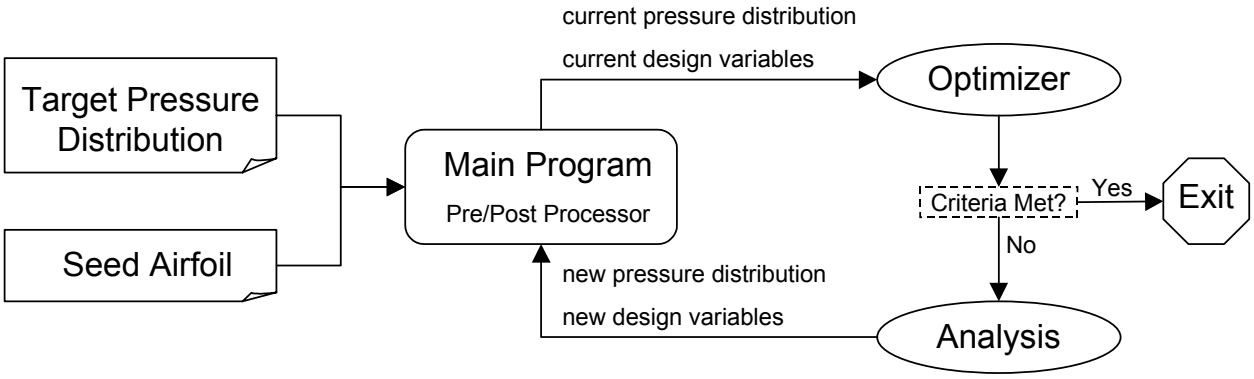

Figure 5. *Airfoil Design Process Flow*

The specific optimizer used for this problem was the `fmincon` function [11] built into Matlab. This is a path-building optimization technique that can handle a nonlinear objective function and nonlinear constraints. It uses sequential quadratic programming (SQP) to reduce each optimizer iteration to a quadratic subproblem that can be easily minimized. This optimizer is quite robust and efficient and therefore is well suited for this design problem. More information on SQP techniques can be found in [12].

6.2 Design Problem Statement

The standard form of the optimization problem statement is as follows:

Minimize:

$$\sum_{i=1}^{N}\left(c_{p,design_i} - c_{p,acutal_i}\right)^2 \tag{22}$$

Subject to:

$$y_{upper_i} \geq y_{lower_i} \tag{23}$$

where $c_{p,design\_i}$ is the design pressure coefficient, $c_{p,actual\_i}$ is the actual pressure coefficient, y_{upper_i} is the y-coordinate of the upper surface, and y_{lower_i} is the y-coordinate of the lower surface, all at the i^{th} chordwise location. The constraints are used such that the upper and lower surfaces do not cross over.

6.3 Code Structure

The design code was written entirely in Matlab. It consisted of eight M-files: one main program and seven custom-written functions. A brief description of each file follows. Any numeric character following the name of these files denotes a revision; hence, `afd3.m` indicates the third major revision to the main airfoil design program. The source code can be found in Appendix A.

`afd3.m`

This served as the main airfoil design routine. It called the appropriate functions to set up the optimization problem, called the optimizer, processed the results, and saved the results to text files.

`afobj3.m`

This function evaluated the objective function for the optimizer based on the current design variable settings. It contained the necessary routines to translate the design variables into an airfoil geometry, run the panel code analysis, and determine the objective function value for the design problem.

`afcon.m`

This function evaluated the constraints for the optimization problem. It read in the airfoil geometry and returned the value of each constraint determined at hard-coded locations. In this manner, it was possible to keep the design program from placing the lower surface above the upper surface while the optimizer explored the design space.

`bump3.m`

This function returned perturbation values for a set of bump functions. Several bump functions were tested in this project as outlined in the next section. The only required input were the x/c stations used to define the airfoil.

`cptarget.m`

This file was used to create the pressure distribution target used by the optimization routine. It read in a tab-delimited text file containing the chordwise location and value of the pressure coefficient over the entire airfoil. As the pressure distribution may not be defined at the same x/c values as the seed airfoil, this function also read in the x/c values of the seed airfoil and used a piecewise cubic Hermite interpolating polynomial (PCHIP) routine to find the target pressure coefficients at these locations.

`panelB.m`

This was a stripped-down version of the Matlab-based panel code provided in reference [8]. The main changes were to take out the pre- and post-processing of results provided in the original code. This function was the most time-intensive of the process and was called by `afobj3.m`, which was in turn called by the optimizer. It provided a pressure distribution and aerodynamic coefficients for an input airfoil geometry.

`process3.m`
This function was virtually identical to `afobj3.m`. It was used to bring several variables stored locally in the memory of the objective function back out for post-processing once the optimizer had converged to a solution, and therefore was called only once at the end of the routine.

`seed.m`
This function read in a tab-delimited text file containing airfoil coordinates. This served as the seed airfoil to start the optimization process, and also determined the number of panels, and thus the speed and accuracy of the panel code.

7. Validation of Design Method

Before the optimization method devised could be used on any pressure distribution, it was necessary to validate the method by inputting a pressure distribution from a known airfoil into the problem. If the optimizer converged to the correct airfoil geometry, then the routine should work. A test of the robustness of the optimizer helps as well, which can be accomplished by giving the optimizer several different starting points.

The target pressure distribution fed into the problem was actually the output of the Matlab panel code run on a NACA 4412 airfoil. This ensured that the differences between the optimized geometry to the pressure distribution were not due to noise between the target data and the method used to generate the pressure distribution.

The design variables used ultimately should be the points used to design the airfoil surface. Therefore, a study was undertaken to see the resulting pressure distribution for a NACA 4412 airfoil for 46, 56, and 141 points. The resulting pressure distribution is shown in Figure 6.

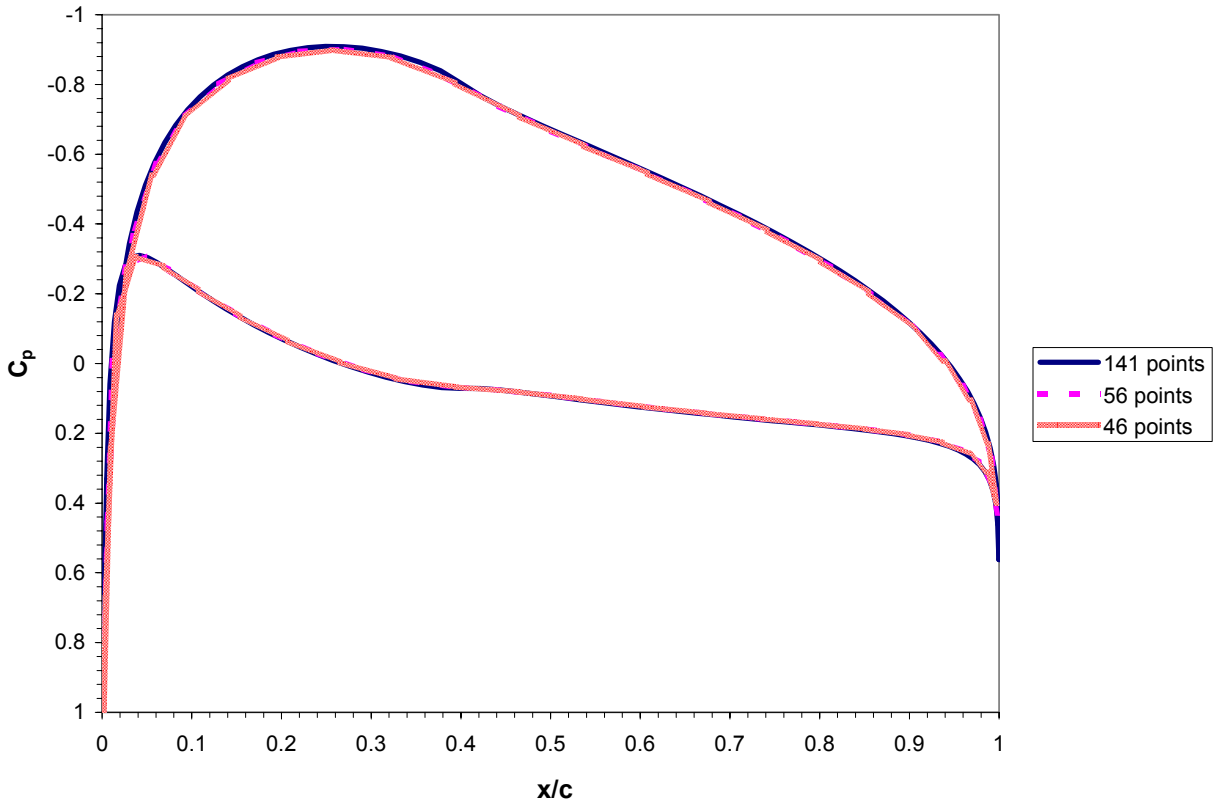

Figure 6. *Effect of varying the number of points on output pressure distribution*

By taking a close look at the 46-point results, one can see that the distribution is still a bit jagged, especially at the lower surface of the leading edge. However, the 56- and 141-point solutions are reasonably smooth. Furthermore, the analysis times were approximately 0.16 seconds, 0.21 seconds, and 1.29 seconds for the 46, 56, and 141-point cases, respectively. Therefore, the 56-point solution offered the best mix of fidelity and speed for application to optimization.

7.1 Bump Functions Tested

Unfortunately, 56 individual points as design variables are quite a few for optimization purposes. Therefore, bump functions can be used instead to provide the answer with fewer degrees of freedom and therefore pose a less cumbersome operation for the optimizer. These bump functions can take many forms, and four were investigated for the validation cases. The design variables came in the form of multipliers to these functions. Two additional design variables were added that acted as scalar multiples to the upper and lower y-ordinates to account for airfoils of similar families with different thicknesses.

The first set of bump functions was a series of several polynomials (denoted *poly1* from here on). They were

$$
\begin{aligned}
p_1(x) &= x^2 \\
p_2(x) &= x^3 \\
p_3(x) &= x^4 \\
p_4(x) &= x^5 \\
p_5(x) &= x^6
\end{aligned}
\tag{24}
$$

where $p_n(x)$ is the bump function, and x is an x/c location from zero to one. When applied to the upper and lower surface, the total number of design variables with these functions is 12 (five upper surface multipliers plus five lower plus two scale factors). A plot of these functions is shown below in Figure 7.

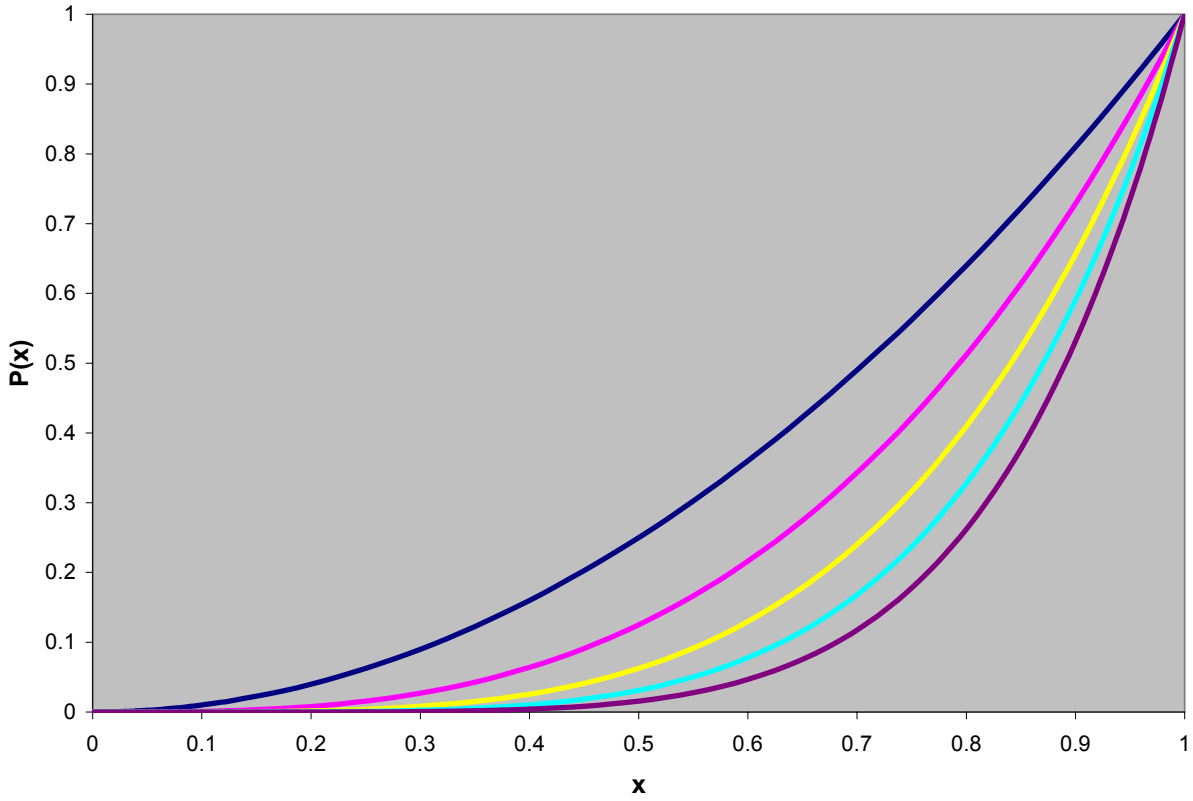

Figure 7. *Bump Functions for poly1*

This set of functions is desirable because it has only a few design variables, reducing the problem from 56 degrees of freedom for the individual airfoil control points to a mere 12. However, these polynomials do not give much authority to change coordinates at the front of the airfoil. As such, another set of polynomial bump functions was investigated that included the polynomials from equation (24) in addition to six more fractional-power terms. These were

$$
\begin{aligned}
p_6(x) &= x^{1/2} \\
p_7(x) &= x^{1/3} \\
p_8(x) &= x^{1/4} \\
p_9(x) &= x^{1/5} \\
p_{10}(x) &= x^{1/6} \\
p_{11}(x) &= x^{1/12}
\end{aligned}
\tag{25}
$$

These functions were referred to as *poly2* and are plotted in Figure 8. Note that this increased to the total number of design variables from 12 to 24, but allowed greater authority to change the airfoil near the leading edge.

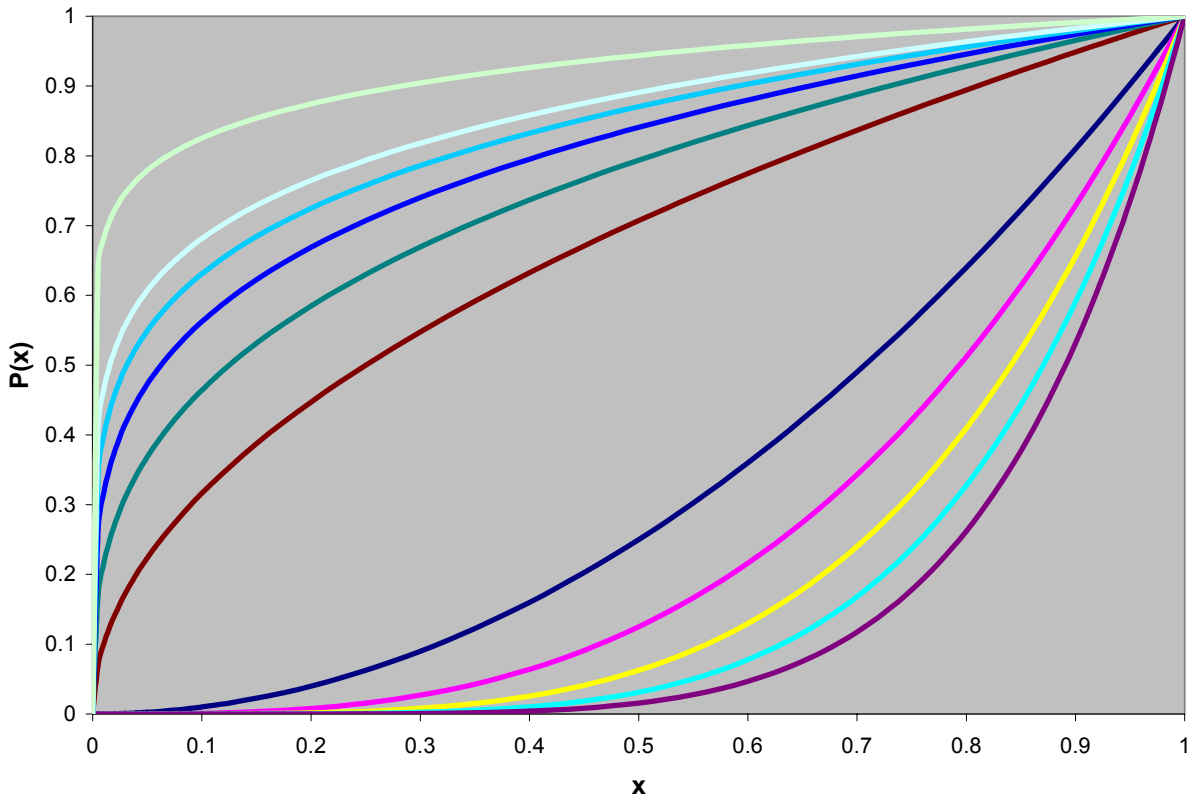

Figure 8. *Bump Functions for poly2*

A third set of hybrid polynomial bump functions (*poly3*) was examined as well. These were defined as

$$
\begin{aligned}
p_1(x) &= -4\left(x^{0.2} - 0.5\right)^2 + 1 \\
p_2(x) &= -4\left(x^{0.3} - 0.5\right)^2 + 1 \\
p_3(x) &= -4\left(x^{0.4} - 0.5\right)^2 + 1 \\
p_4(x) &= -4\left(x^{0.5} - 0.5\right)^2 + 1 \\
p_5(x) &= -4\left(x^{0.7} - 0.5\right)^2 + 1 \\
p_6(x) &= -4\left(x - 0.5\right)^2 + 1 \\
p_7(x) &= -4\left(x^2 - 0.5\right)^2 + 1 \\
p_8(x) &= -4\left(x^3 - 0.5\right)^2 + 1 \\
p_9(x) &= -4\left(x^4 - 0.5\right)^2 + 1 \\
p_{10}(x) &= -4\left(x^6 - 0.5\right)^2 + 1 \\
p_{11}(x) &= -4\left(x^{12} - 0.5\right)^2 + 1
\end{aligned}
\tag{26}
$$

Again, this resulted in 24 total design variables. The functions are plotted in Figure 9.

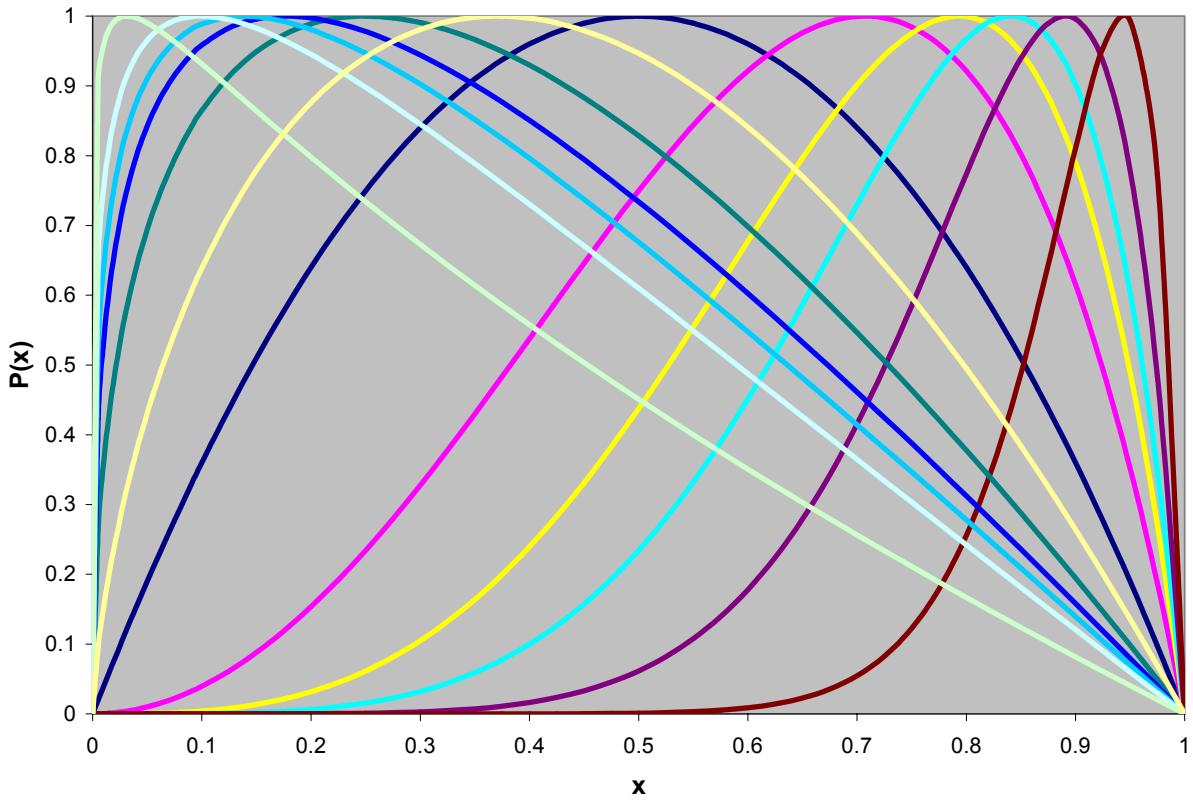

Figure 9. *Bump Functions for poly3*

One final set of bump functions was taken from reference [13]. This used several Hicks-Henne functions which were either sinusoidal or polynomial in nature. They are given by

$$
\begin{aligned}
p_1(x) &= \frac{10(1-x)x^{a_{12}}}{e^{20x}} \\
p_2(x) &= \frac{10(1-x)x^{a_{12}}}{e^{40x}} \\
p_3(x) &= \frac{\sqrt{x}(1-x)}{e^{3x}} \\
p_4(x) &= \sin^5\left(\pi x^{0.5757166}\right) \\
p_5(x) &= \sin^5\left(\pi x^{0.7564708}\right) \\
p_6(x) &= \sin^5(\pi x) \\
p_7(x) &= \sin^5\left(\pi x^{1.356915}\right) \\
p_8(x) &= \sin^5\left(\pi x^{1.943358}\right) \\
p_9(x) &= \sin^5\left(\pi x^{3.106283}\right) \\
p_{10}(x) &= \sin^5\left(\pi x^{6.578813}\right) \\
p_{11}(x) &= x^{10}
\end{aligned}
\tag{27}
$$

where a_{12} is the 12th design variable (the other 11 for each surface being scalar multiples to each bump function). These functions were not combined with airfoil scale factors, so this once again results in 24 design variables. These functions are denoted *hicks* from here on and are shown in Figure 10.

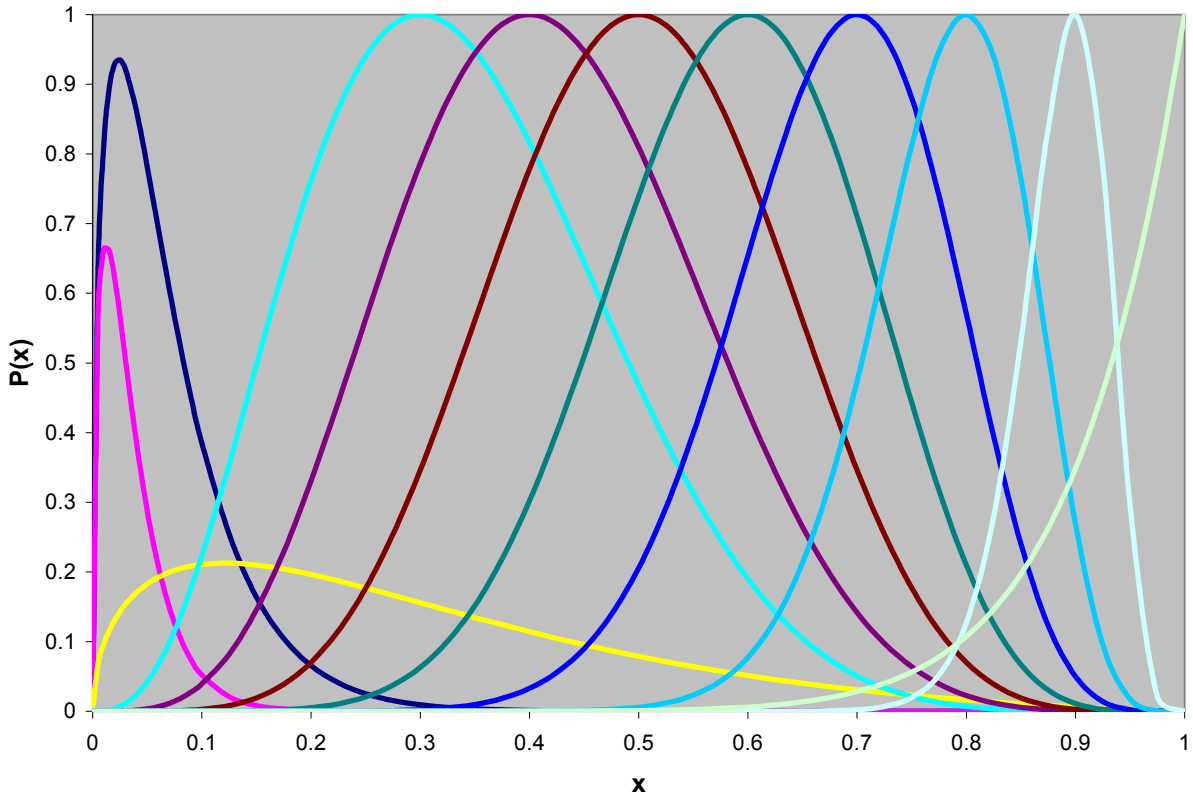

Figure 10. *Bump Functions for hicks*

7.2 Seed Airfoils for Validation Problem

The optimization process requires a seed airfoil to work from. This airfoil defines the starting geometry for the optimization process as well as the number of points. It is also important in the case of multimodal problems, as SQP methods will only find local minimums near the starting point.

Two airfoils were chosen as seed airfoils. The first was a NACA 0012, a rather generic symmetric airfoil typically used in such studies. However, this is a NACA four-series airfoil, and thus has the same thickness distribution (and indeed the same thickness) as the NACA 4412 airfoil the program is supposed to design. Therefore, checking with an airfoil from a different family with a different thickness distribution would be a good measure of the robustness of the solution. As such, the NACA 23015 airfoil was selected as another seed airfoil to act as a check on the solution from the 0012 airfoil.

7.3 Results

The experimental setup involved two seed airfoils with four different sets of bump functions for a total of eight test cases. The results that follow are primarily from the NACA 0012 seed airfoil.

The geometry that resulted from all four bump functions for the NACA 0012 seed can be seen in Figure 11. The results are plotted against the actual NACA 4412 airfoil that is the target result.

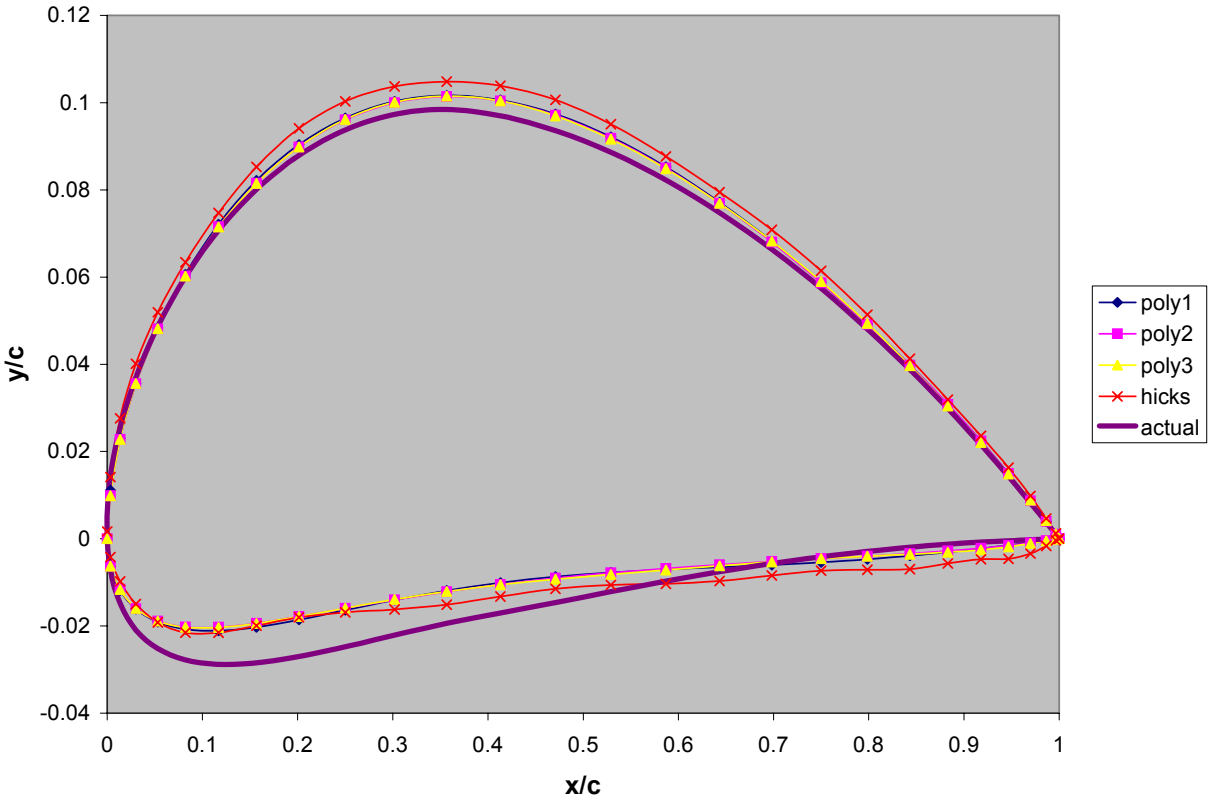

Figure 11. *Geometries generated from four bump functions compared to NACA 4412*

This is rather interesting, as is shows that all four methods converged to almost the same shape. The converged shape is close to a NACA 4412 airfoil in all respects except for the region around the lower surface of the leading edge. Of all of the bump functions, the three polynomial functions do not appear to create any better result than the other, but the modified Hicks-Henne functions seem to have some issues as the upper surface is slightly too high and entire lower surface is off. The pressure distribution, shown in Figure 12, tells the same story from a slightly different perspective.

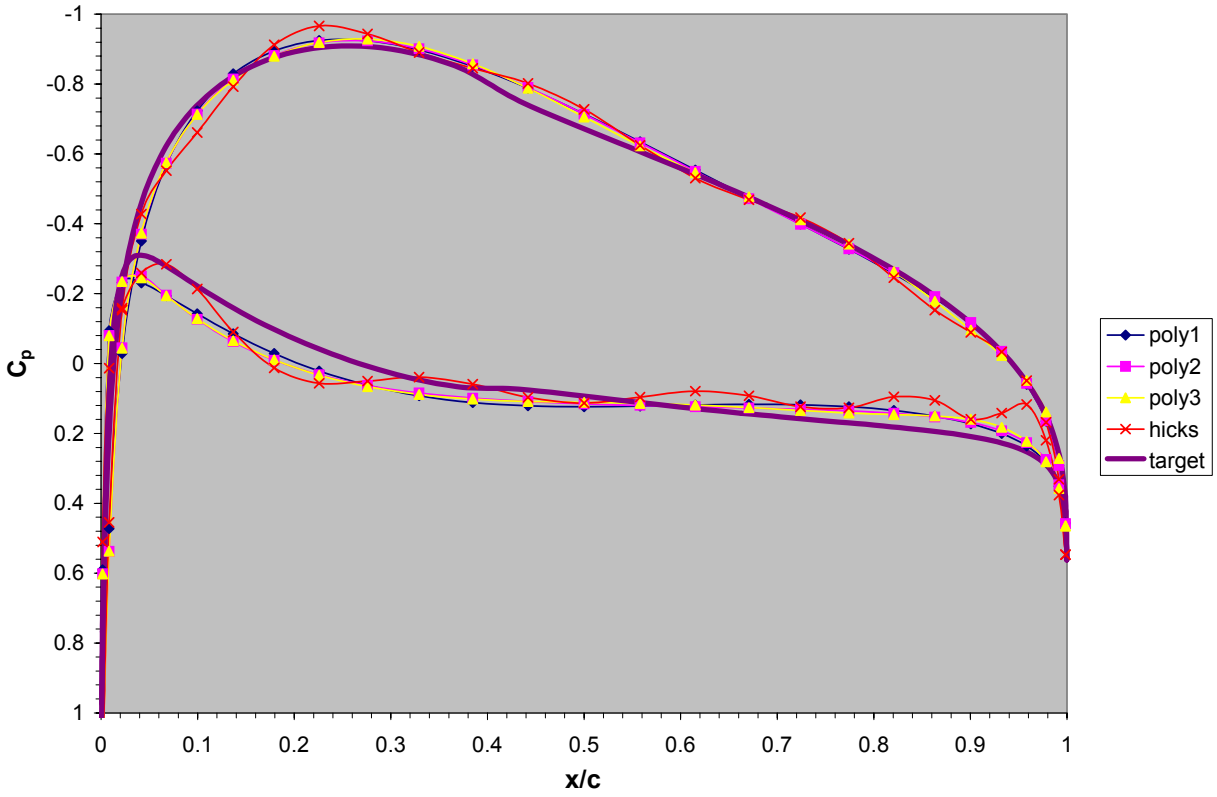

Figure 12. *Pressure distribution outputs compared to target pressure distribution*

Here, the pressure distribution near the lower surface of the leading edge is too high, while it drops slightly below specification near the trailing edge. The upper surface appears to be much better, which only a slight discrepancy near the point of maximum thickness. Again, the Hicks-Henne functions appear to have the most issues meeting the target pressure distribution. This could be due to the sinusoidal nature of these functions, as the output pressure distribution has a moderate wave form.

7.4 Convergence

If all of the bump functions generate the same geometry, which is the most efficient? Which is the most robust? These are questions that need to be answered before a single set of functions can be chosen. Therefore, the convergence history of each set of bump functions was tracked as well. The number of optimizer iterations for SQP is not a good measure of efficiency; in all likelihood, a set of functions with more design variables will require fewer optimizer iterations even though the optimization problem itself is much harder. A much better measure of efficiency is the number of times the objective function is called. Each call represents one run of the panel code, which amounts to approximately a quarter of a second. Figure 13 gives the convergence history of each bump function from the NACA 0012 seed. Each point represents one SQP optimizer iteration.

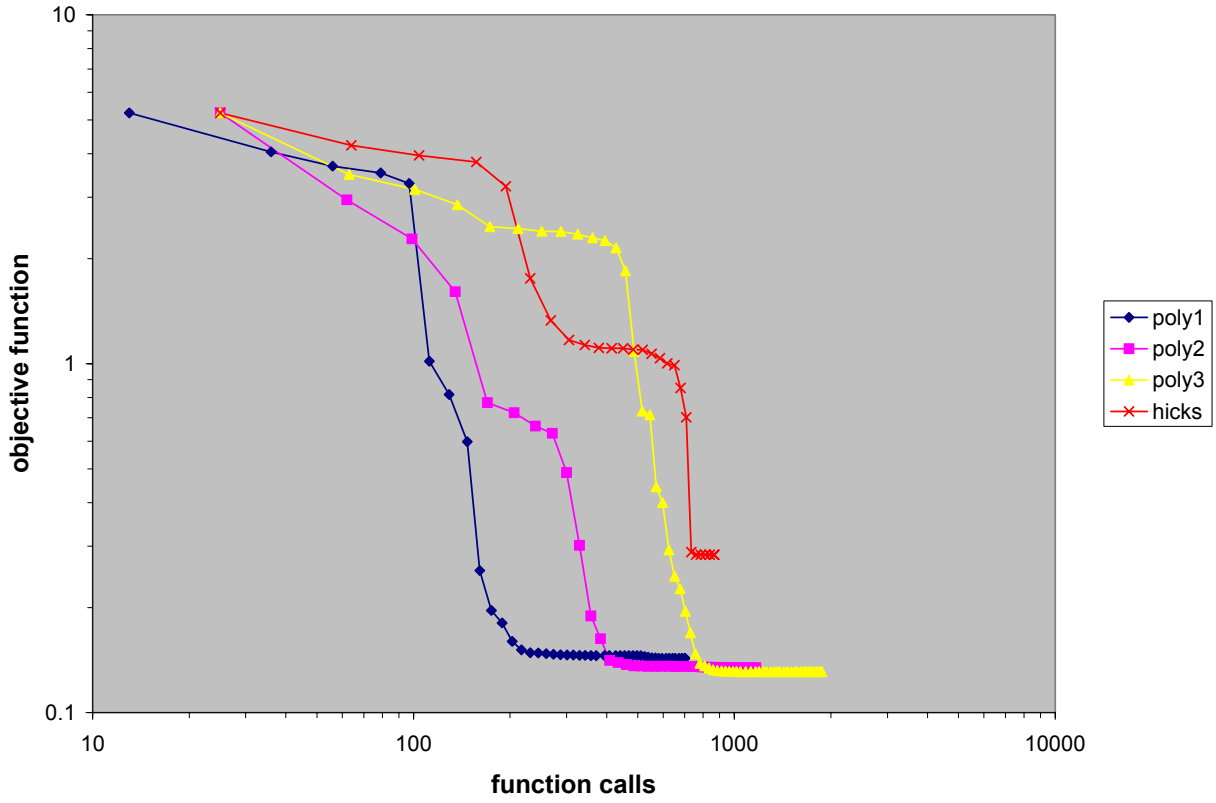

Figure 13. *Objective function history for each set of bump functions*

Here, it appears that *poly1* converges the fastest, while *poly3* gets the lowest objective function value. Furthermore, *hicks* converges faster than *poly2* or *poly3*, but arrives at a much higher value of the objective function, so therefore it is not reliably converged.

Figure 14, given on the next page, gives the resulting geometries from the NACA 23015 seed. These geometries are very similar to those outlined in Figure 11, thus indicating that these four sets of bump functions are relatively robust. Not shown are the pressure distributions or convergence histories, but these are included in the mid-term presentation. In general, the convergence took a bit longer and the values were slightly worse. Overall, however, the results were comparable.

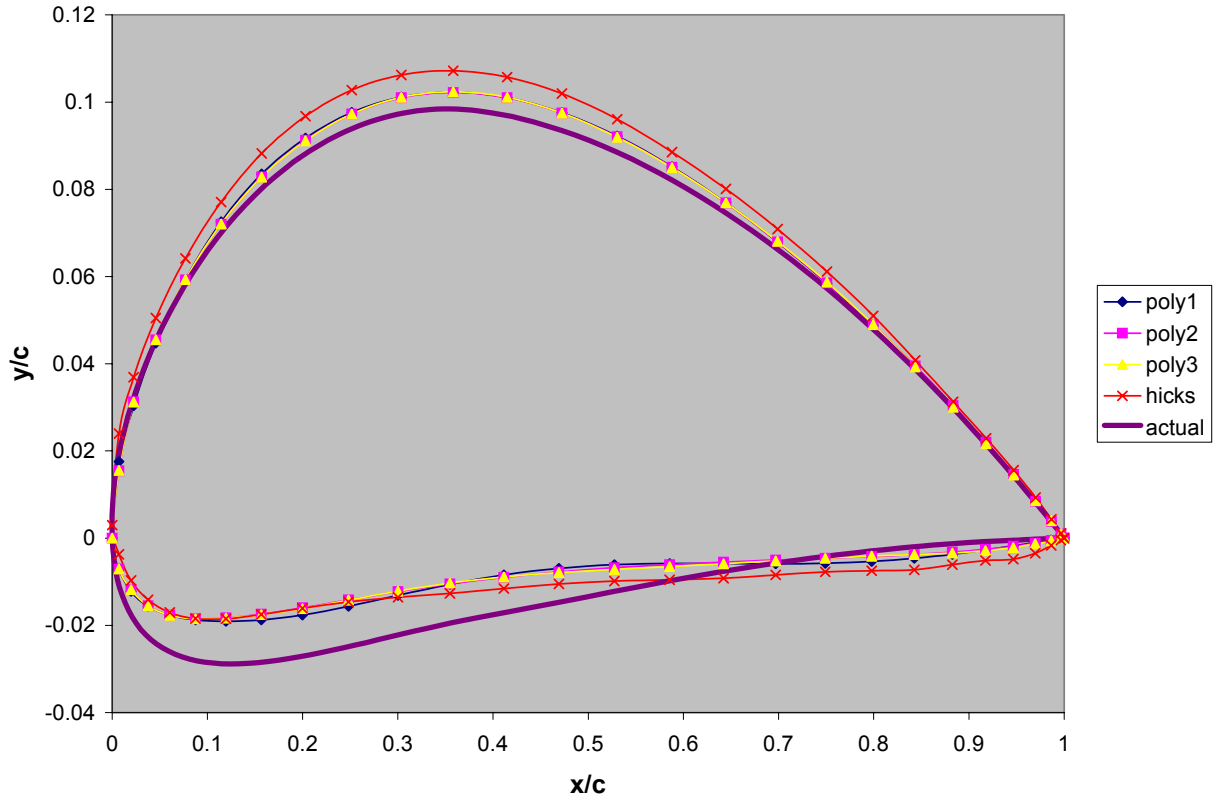

Figure 14. *Geometries generated from NACA 23015 seed*

These results seem to indicate that *poly3* is the best solution for robustness while *poly1* is the best for efficiency. Efficiency could be further increased with an executable version of the analysis code. One lingering problem is related to the final geometries – why did each airfoil converge to the wrong shape, albeit slightly? This indicated that there were a few bugs left in the analysis and optimization codes that needed to be worked out. Indeed, these bugs were identified and fixed, and the final geometry resulting from *poly3* ended up matching the 4412 geometry almost exactly for both the 0012 and 23015 seeds. Unfortunately, due to time constraints, the validation tests for all four bump function cases could not be rerun (which is why the old results were presented in this section).

8. Design Studies

At this point, it is possible to begin airfoil design studies. This involves the selection of a target pressure distribution and to be input into the design code. Unfortunately, defining a target pressure distribution is as much an art as it is a science. This process often involves iteration between defining the pressure distribution and running the code, since some pressure distributions may be physically impossible or cause problems with convergence.

8.1 Input Pressure Distributions

Selection of a target pressure distribution depends greatly on the application. This project focuses on low-Reynolds number airfoil design with specific application to a flying wing configuration. This configuration requirement forces the designer to use reflex camber or a combination of wing sweep and twist (geometric or aerodynamic) to trim the aircraft in level flight. Furthermore, pressure distributions can be tailored such that the airfoil has desirable stall characteristics, low drag rise at high speeds, promotion of laminar flow, and so on. This can be done through tailoring the adverse pressure gradient on the upper surface, changing the leading edge radius, and a variety of other means.

The pressure distributions for this project were created via polynomials of varying degrees. Specifically, they were of the form

$$a\left(x^b - c\right)^d + e \tag{28}$$

where x is the normalized chordwise distance from the leading edge, and a, b, c, d, and e are shape parameters. Manipulation of these five parameters yielded a plethora of options for defining the pressure distributions on the lower and upper surfaces. Furthermore, use of a polynomial ensured that the pressure distribution was smooth and continuous, something quite important because the airfoil design program used cubic interpolation between the specified pressure distribution points. A spreadsheet program with built-in numerical solving capability was used to define the pressure distributions for a specified section lift coefficient (C_l) and moment coefficient (C_m).

A section design lift coefficient of 0.6 was initially specified for the design cases. Two pressure distributions were defined for this lift coefficient. The first was a relatively flat or "rooftop" pressure distribution that would promote laminar flow over the upper surface, taking advantage of the low Reynolds number effects. However, this pressure distribution is highly rear-loading, indicating an excessively negative moment coefficient. This is not ideal as it imposes more trim drag on an airframe. It is also highly undesirable for flying wing aircraft, as they cannot use horizontal tail surfaces to provide trim. Therefore, a second pressure distribution was defined such that the design lift coefficient of 0.6 could be delivered with a zero section moment coefficient. This is accomplished by heavy front-loading of the airfoil combined with a downward pressure loading near the trailing edge. These airfoils are often referred to as reflex camber airfoils because of the negative camber near the trailing edge. From here on, the first pressure distribution / airfoil is referred to as *rooftop*, and the second as *reflex*. A plot of both of these pressure distributions is shown in Figure 15.

These cases differed from the validation cases in that the angle of attack was allowed to vary from –5.0 to 5.0 degrees as the 25th design variable. This allowed the design program to place the leading edge and trailing edge perfectly along the chord line, and reflected the design angle of attack (incidence) normally associated with airfoils.

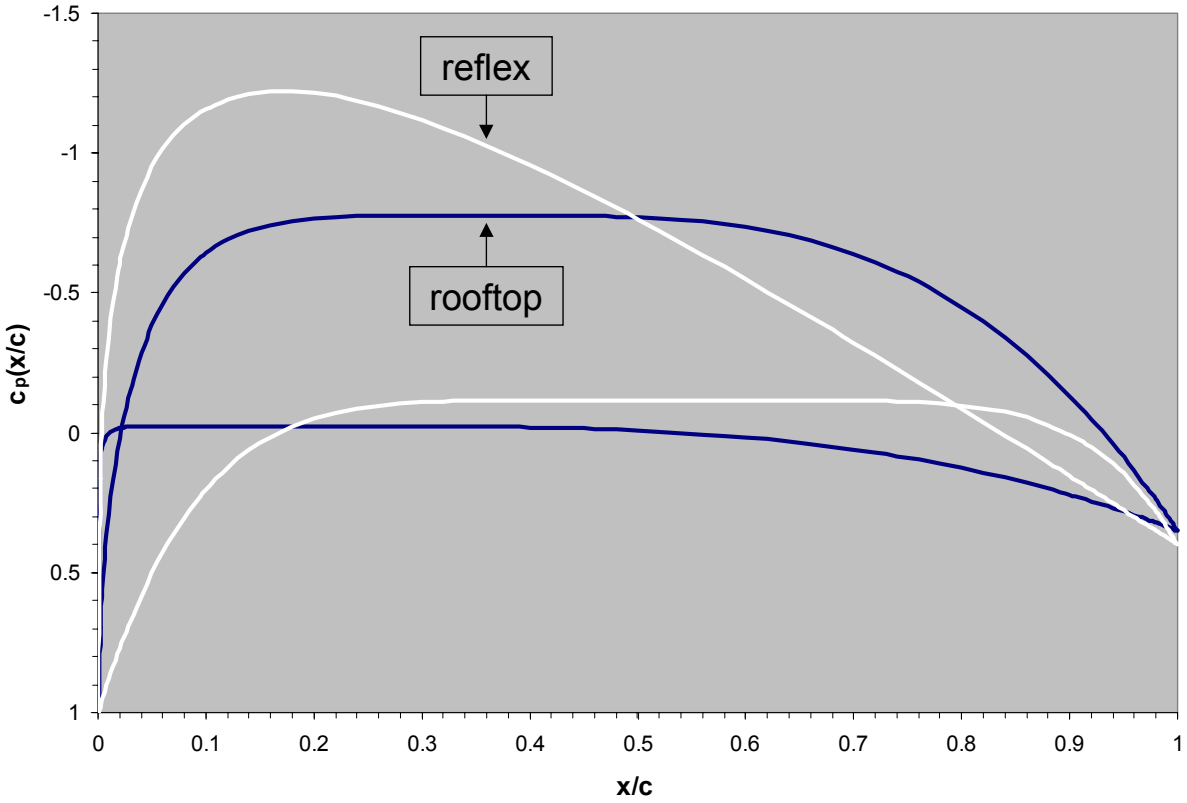

Figure 15. *Input Pressure Distributions for Design Case, C_l = 0.6*

8.2 Results and Comparison to XFOIL

Each case was run with these input pressure distributions, and both converged quite well. These results, however, were for incompressible, inviscid flow, so they were compared against output from XFOIL for an inviscid case (to double-check the Matlab panel code) and a viscous case for a Reynolds number of 1,000,000. Table 1 compares the results for the target, converged, XFOIL inviscid, and XFOIL viscous solutions. The resulting pressure distributions for the *rooftop* and *reflex* airfoils are given in Figures 16 and 17.

		Matlab		*XFOIL*	
	parameter	*target*	*converged*	*inviscid*	*viscous*
rooftop	C_l	0.6000	0.6066	0.6093	0.5322
	C_m	*	-0.1387	-0.1394	-0.1225
	C_d	0.0000	0.0008	-0.0003	0.0055
reflex	C_l	0.6000	0.6169	0.6153	0.5637
	C_m	0.0000	-0.0043	-0.0078	0.0027
	C_d	0.0000	0.0008	-0.0002	0.0081

Table 1. *Comparison of Panel Code Results*

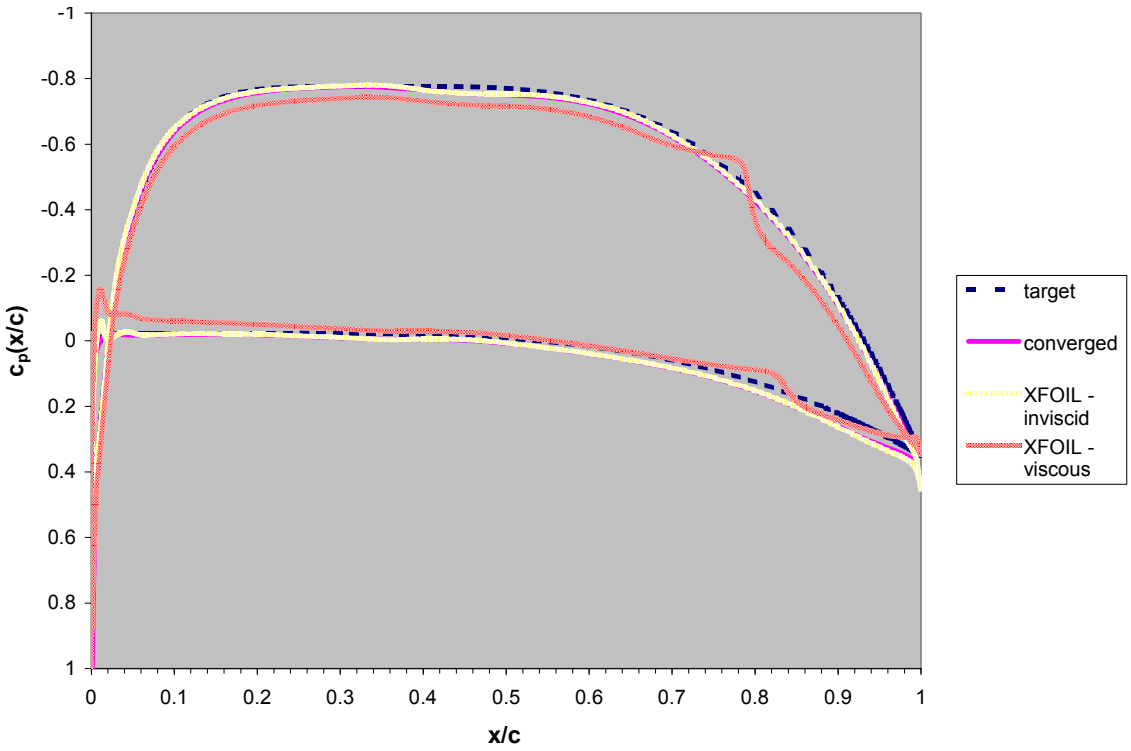

Figure 16. *Pressure Distribution Comparison for Rooftop Pressure Distribution*

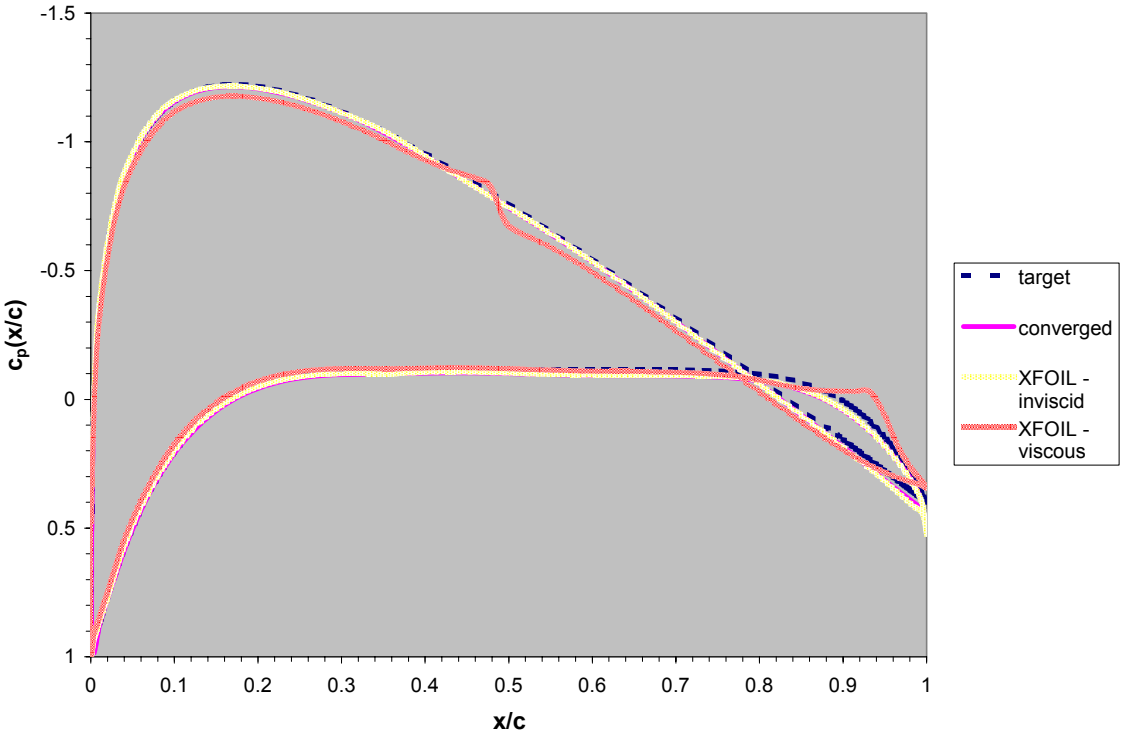

Figure 17. *Pressure Distribution Comparison for Reflex Pressure Distribution*

Notice that the XFOIL inviscid solution falls almost exactly on the converged inviscid solution. Also, both inviscid solutions are very close to the target, indicating that the final geometry converged well. The viscous solution is reasonably close, but, as seen from Table 1, the lift coefficient drops by several hundredths. This is because XFOIL accounts for the displacement thickness of the boundary layer when computing the viscous solution, which alters the panel code results accordingly. As such, the results from the *rooftop* airfoil show a reduction is lift and nose-down moment because, as a rear-loading airfoil, the displacement thickness near the trailing edge is larger, therefore reducing the effective camber in that region. However, the *reflex* airfoil, while also suffering less lift in the viscous solution, is less effected because more of its lift comes from the forward portion of the airfoil.

The geometry of the two airfoils, shown with both axes roughly to scale, is shown in Figure 18 below. Notice that the *reflex* airfoil is slightly thicker and displays a slight amount of reflex camber near the trailing edge. The rooftop airfoil, while highly cambered, has a relatively evenly distributed thickness form.

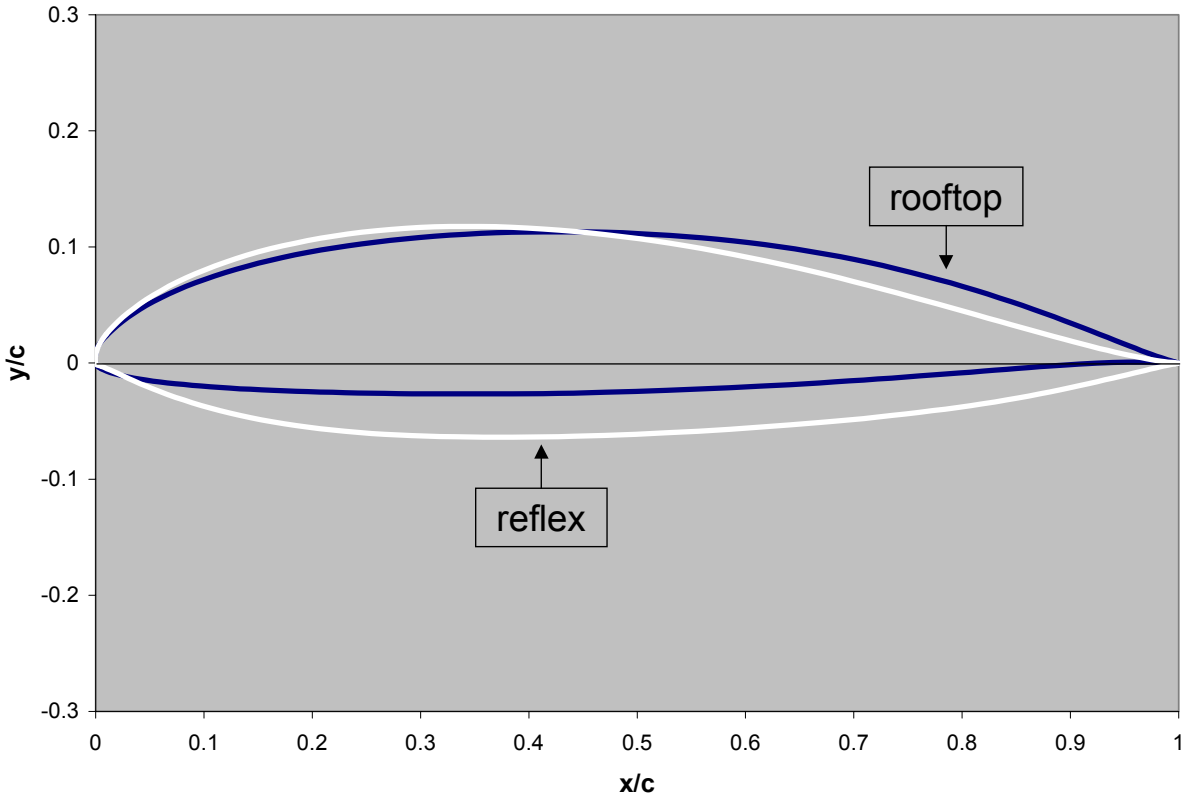

Figure 18. Rooftop and Reflex Airfoil Geometries

The performance of these airfoils can be summarized in three charts: C_d versus C_l (called a drag polar), C_l versus angle of attack (α), and C_m versus α. These can be found in Figures 19 through 21, and were all generated from viscous XFOIL solutions.

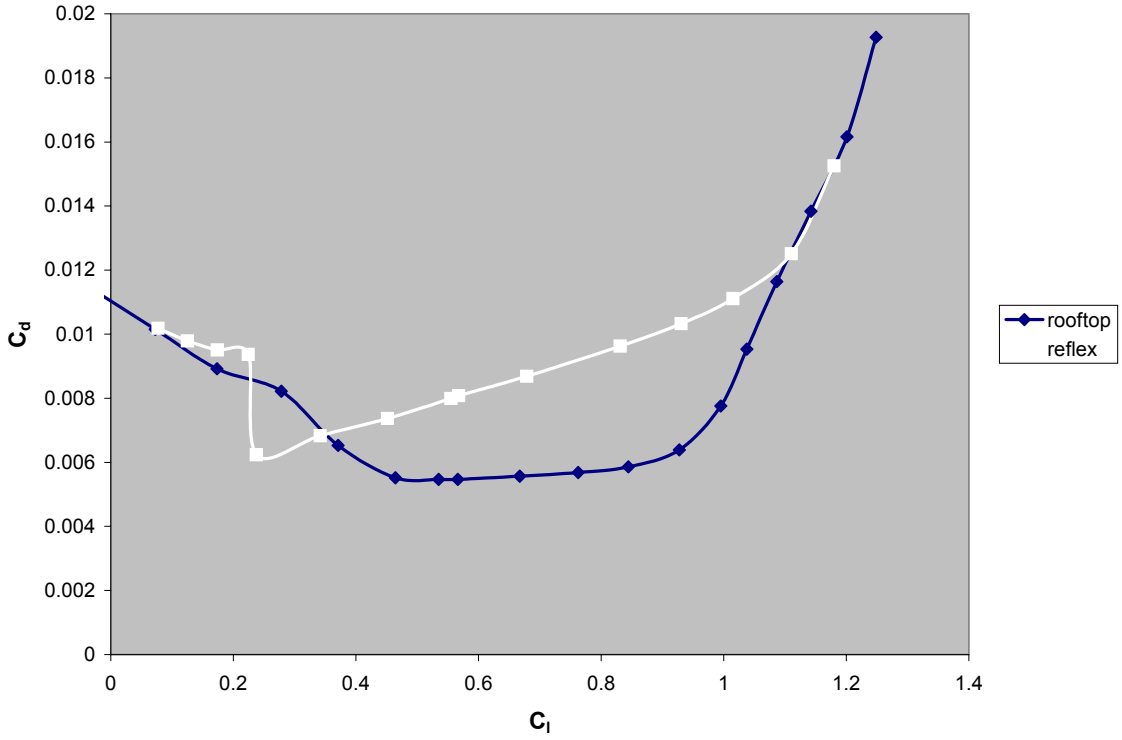

Figure 19. Drag Polar for Rooftop and Reflex Airfoils

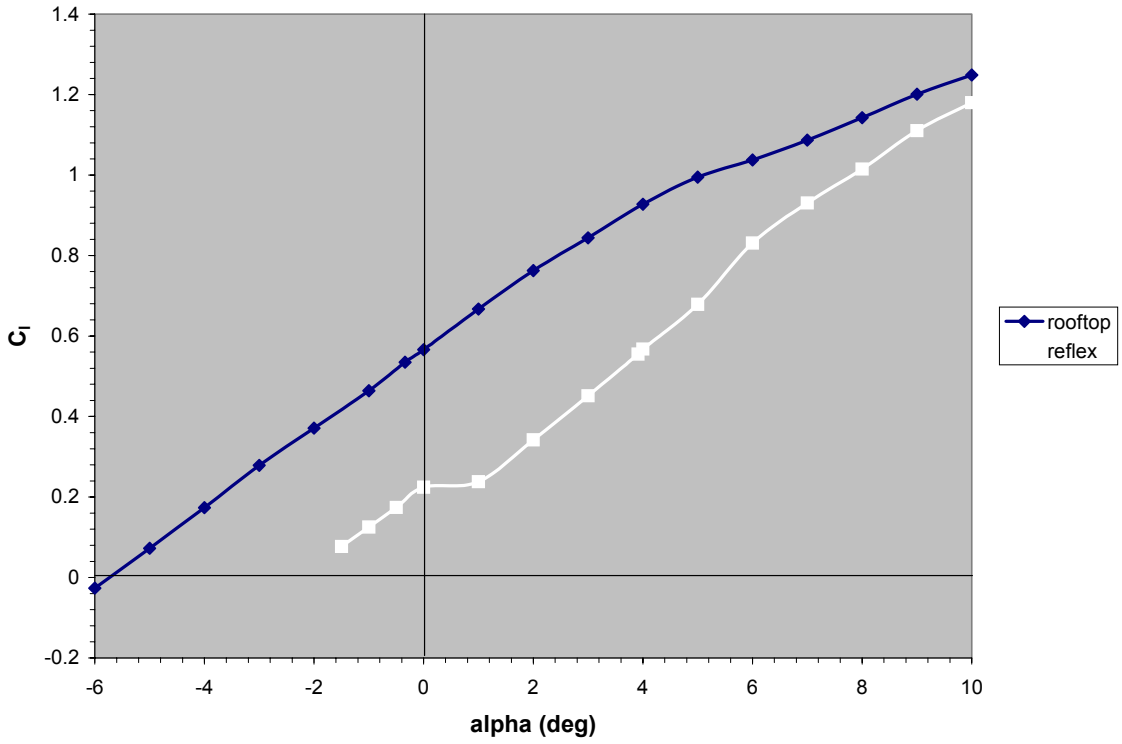

Figure 20. Lift-Curve for Rooftop and Reflex Airfoils

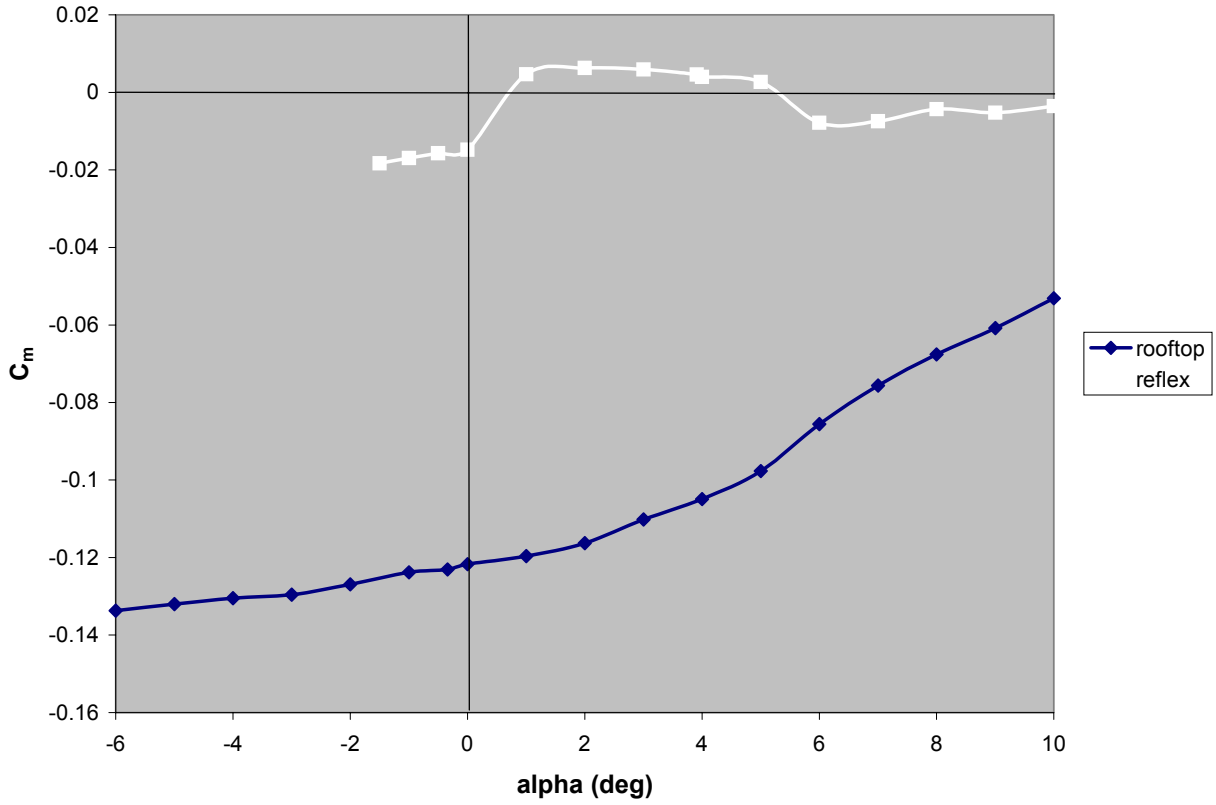

Figure 21. Moment-Curve for Rooftop and Reflex Airfoils

The drag polar seems to indicate that the *rooftop* airfoil experiences laminar flow between a C_l of 0.4 and 1.0 as seen by the drag bucket. Outside this region, the flow becomes turbulent because the pressure gradient becomes too adverse. Furthermore, notice that this region lies around the design lift coefficient of 0.6, which is highly desirable for low drag during cruise.

The drag polar of the *reflex* airfoil is quite odd as it has a sudden decrease in drag at a C_l of approximately 0.2. This is again likely due to boundary layer turbulence or perhaps even separation. Furthermore, it is highly likely that the reflex portion of the airfoil has a heavier download and therefore is prone to adverse pressure gradients at low lift coefficients.

Finally, it is interesting to see the behavior of the moment coefficients. The rooftop airfoil, as predicted, is highly rear-loaded and consequently has a stronger nose-down pitching moment. The reflex airfoil, as hoped, has a very small moment coefficient that can most likely be trimmed out with elevon deflection.

9. Future Studies

It appears that the airfoil design program developed over the course of this project is capable of designing airfoils with unorthodox but physically possible pressure distributions. Its capability could be increased, however, if it were tied to a boundary layer analysis as used in XFOIL. This would enable the design program to tailor the aircraft for favorable transition regions and to delay stall as well as better meet the prescribed pressure distributions.

This routine could be integrated with a three-dimensional routine to handle airfoil design for optimum performance of a given planform. The objective would most likely be to minimize induced drag by prescribing an elliptic spanwise lift distribution. If the required wing lift coefficient were known, a 2-D airfoil design code could design the airfoils at each station and a 3-D analysis code could define the resulting spanwise lift distribution. These tools could iterate until the solution converged to an elliptic spanwise distribution.

References

1. Abbott, I. H., and von Doenhoff, A. E., *Theory of Wing Sections*, Dover, New York, 1959.
2. "UIUC Airfoil Coordinates Database," <http://amber.aae.uiuc.edu/~m-selig/ads/coord_database.html>
3. "Cessna/ONR/AIAA 2002-2003 Design / Build / Fly Competition Home Page," <http://amber.aae.uiuc.edu/~aiaadbf/>
4. Mattingly, J. D., Heiser, W. H., Daley, D. H., *Aircraft Engine Design*, AIAA, New York, 1987.
5. *Aerodynamics for Naval Aviators*, Naval Air Systems Command, 1965.
6. Anderson, John D., *Fundamentals of Aerodynamics, 2nd Edition*, McGraw-Hill, New York, 1991.
7. Katz, Joseph, and Plotkin, Allen, *Low Speed Aerodynamics: From Wing Theory to Panel Methods*, McGraw-Hill, New York, 1991.
8. "AE 3903 / 4903 Course Web Page," <http://www.ae.gatech.edu/~lsankar/AE3903>.
9. Drela, Mark, *XFOIL 6.94 User Guide*, Massachusetts Institute of Technology, Cambridge, Massachusetts, December 2001.
10. Drela, Mark, "XFOIL: An Analysis and Design System for Low Reynolds Number Airfoils," *Conference Proceedings on Low Reynolds Number Aerodynamics*, Notre Dame, Indiana, 5-7 June 1989.
11. *Matlab Release 12.1 User's Guide*, The MathWorks, Inc., Natick, Massachusetts
12. Vanderplaats, Garret N., *Numerical Techniques for Engineering Design*, Vanderplaats Research & Development, Inc., Colorado Springs, Colorado, 1999.
13. Lores, M. E., Burdges, K. P., Shrewsbury, G. D., "Analysis of a Theoretically Optimized Transonic Airfoil," NASA Contractor Report 3065, November 1978.

Appendix A:
Source Code

afd3.m

```
% afd3.m
% this is the main airfoil design code

clear all;

% start timer
tic;

global y;

% read in seed airfoil and get data
% the dash number after the airfoil indicates how many points are used to define the shape
airfoil = 'naca4412-056.txt';
[xu,xL,yu0,yL0,nu,nL] = seed(airfoil);

% specify angle of attack and Mach number
alpha = 0.0;
M = 0.0;

% initialize design variables
desVar = zeros(25,1);
numVars = length(desVar);
sfLoc = numVars/2 - 0.5;
desVar(sfLoc,1) = 1;
desVar(2*sfLoc,1) = 1;
desVar(numVars,1) = alpha;

% find cptarget and prepare to pass it to the objective function
[junk,cpt] = cptarget(xu,xL);

% add empty values to pass to optimizer
A = [];
b = [];
Aeq = [];
beq = [];

% specify upper and lower bounds
LBs = ones(numVars,1);
UBs = ones(numVars,1);
LB = -10.*LBs;
UB = 10.*LBs;
LB(sfLoc,1) = 0.001;
LB(2*sfLoc,1) = 0.001;
UB(numVars,1) = 5;
LB(numVars,1) = -5;

% specify starting point for design variables
x0 = desVar;

% optimizer options
options = [];
options =
optimset('Display','iter','MaxFunEvals',50000,'LargeScale','off','MaxIter',2500,'TolFun',1E-
6,'TolCon', 1E-6);

% optimizer call
[desVar,fval,exitflag,output] =
fmincon(@afobj3,x0,A,b,Aeq,beq,LB,UB,@afcon,options,xu,xL,yu0,yL0,M,nu,nL,cpt);

% stop timer
toc;
t = toc;

% post-process results
alpha = desVar(numVars);
[x,y,cl,cm,cd,cpc,xmid] = process3(desVar,xu,xL,yu0,yL0,nu,nL,M);
r1 = [xmid cpc];
r2 = [x' y'];
```

```
r3 = [alpha cl cm cd t];
dlmwrite('1pressure_dist.txt',r1,'\t');
dlmwrite('2coords.txt',r2,'\t');
dlmwrite('3results.txt',r3,'\t');
dlmwrite('4desvar.txt',desVar,'\t');

% plot options
figure(1);
plot(xmid,cpc,xmid,cpt,'--');
figure(2);
plot(x,y);
axis equal;
```

afobj3.m

```
% afobj3.m
% aifoil design objective function

function obj = afobj2(desVar,xu,xL,yu0,yL0,M,nu,nL,cpt)

global y;

% define alpha
last = length(desVar);
alpha = desVar(last,1);

% compute bump functions
pu = bump3(xu');
pL = bump3(xL');

% compute bump amounts
totVars = length(desVar) - 1;
nVars = totVars/2;
desVaru = zeros(nVars,1);
desVarL = zeros(nVars,1);
desVaru = desVar(1:nVars,1);
desVarL = desVar(nVars+1:totVars,1);
count = nVars - 1;
pertu = zeros(count,nu);
pertL = zeros(count,nL);
for i = 1:count
    pertu(i,:) = pu(i,:).*desVaru(i,1);
    pertL(i,:) = pL(i,:).*desVarL(i,1);
end

dyu = zeros(nu,1);
for j = 1:nu
    dyu(j,1) = sum(pertu(:,j));
end
dyL = zeros(nL,1);
for k = 1:nL
    dyL(k,1) = sum(pertL(:,k));
end

yu = (yu0 + dyu).*desVaru(nVars,1);
yL = (yL0 + dyL).*desVarL(nVars,1);

% stretch y-direction for compressibility correction
corr = xu(nu)./(sqrt(1-M^2));
yu = yu.*corr;
yL = yL.*corr;

% reformat data for panel code
points = nu + nL - 1;
x = zeros(1,points);
y = zeros(1,points);
x(1,nL:points) = xu';
y(1,nL:points) = yu';
for ii = 1:nL
    x(1,nL + 1 - ii) = xL(ii,1);
    y(1,nL + 1 - ii) = yL(ii,1);
end

% run panel code
[cpc,cl,cm,cd,xmid] = panelB(x,y,alpha,nu,nL);

% compare cp results
mat = (cpc - cpt).^2;

% define objective function value
obj = sum(mat);
```

afcon.m

```
% afcon.m
% this file has the constraints for the airfoil design program

function [c,ceq] = afcon(desVar,xu,xL,yu0,yL0,M,nu,nL,cpt)

global y;

% pick stations on the airfoil to enforce the thickness constraint
xtf = [0.1:0.1:0.7];
xtb = [0.8:0.01:0.99];
xtest = [xtf xtb];

% reformat y indices for interpolation purposes
n = nu + nL - 1;
yu = y(1,nL:n);
for i = 1:nL
    yL(1,i) = y(1,nL - i + 1);
end

% define "minimum thickness function"
tmin = -0.04.*(xu - 0.5).^2 + 0.01;

% interpolate y values at the test points along the airfoil
yutest = interp1(xu,yu,xtest,'pchip');
yLtest = interp1(xL,yL,xtest,'pchip');
tmintest = interp1(xu,tmin,xtest,'pchip');

c = yLtest - yutest + tmintest;

ceq = [];
```

bump3.m

```
% bump3.m
% different polynomial bump function

function p = bump3(x)

n = length(x);

% bump functions
p = zeros(11,n);
p(1,:) = -4.*((x.^0.2 - 0.5).^2) + 1;
p(2,:) = -4.*((x.^0.3 - 0.5).^2) + 1;
p(3,:) = -4.*((x.^0.4 - 0.5).^2) + 1;
p(4,:) = -4.*((x.^0.5 - 0.5).^2) + 1;
p(5,:) = -4.*((x.^0.7 - 0.5).^2) + 1;
p(6,:) = -4.*((x - 0.5).^2) + 1;
p(7,:) = -4.*((x.^2 - 0.5).^2) + 1;
p(8,:) = -4.*((x.^3 - 0.5).^2) + 1;
p(9,:) = -4.*((x.^4 - 0.5).^2) + 1;
p(10,:) = -4.*((x.^6 - 0.5).^2) + 1;
p(11,:) = -4.*((x.^12 - 0.5).^2) + 1;
```

cptarget.m

```
% cptarget.m
% this function formats the Cp target data via interpolation

function [xmid,cpt] = cptarget(xu,xL)

% read in target Cp values
cptar = dlmread('cptarget.txt','\t');
ncptu = cptar(1,1);
ncptL = cptar(1,2);
cptLstart = ncptu + 2;
cptLend = ncptu + ncptL + 1;
xtu = cptar(2:ncptu+1,1);
xtL = cptar(cptLstart:cptLend,1);
cptu = cptar(2:ncptu+1,2);
cptL = cptar(cptLstart:cptLend,2);

% find midpoints of input x's and break into upper and lower surfaces
nu = length(xu);
nL = length(xL);
points = nu + nL - 1;
x = zeros(points,1);
x(nL:points,1) = xu;
for i = 1:nL
    x(nL + 1 - i,1) = xL(i,1);
end

nmid = points - 1;
xmid = zeros(nmid,1);
for i = 1:nmid
  xmid(i,1) = 0.5 * (x(i) + x(i+1));
end

nmidu = nu - 1;
nmidL = nL - 1;
xmidu = xmid(nL:nmid,1);
%xmidL = x(1:nmidL);
xmidL = zeros(nmidL,1);
for k = 1:nmidL
    xmidL(nmidL + 1 - k,1) = xmid(k,1);
end

% interpolate target cp values for x's used in airfoil design
cptui = interp1(xtu,cptu,xmidu,'pchip','extrap');
cptLi = interp1(xtL,cptL,xmidL,'pchip','extrap');

% reformat target cp values
cpt = zeros(nmid,1);
cpt(nmidL+1:nmid,1) = cptui;
for j = 1:nmidL
    cpt(nmidL + 1 - j,1) = cptLi(j,1);
end
```

panelB.m

```
% this is a function for use is an airfoil design program
% it is a modified version of Dr. Sankar's panel code

function [cp,cl,cm,cd,xmid] = panelB(x,y,alpha,nu,nl)

% Assemble the Influence Coefficient Matrix A

n = nu + nl - 2;
A = zeros(n+1,n+1);
ds = zeros(1,n);

for i = 1:n
   t1= x(i+1)-x(i);
   t2 = y(i+1)-y(i);
   ds(i) = sqrt(t1*t1+t2*t2);
end

for j = 1:n
    a(j,n+1) = 1.0;
    for i = 1:n
        if i == j
        a(i,i) = ds(i)/(2.*pi) *(log(0.5*ds(i)) - 1.0);
        else
            xm1 = 0.5 * (x(j)+x(j+1));
            ym1 = 0.5 * (y(j)+y(j+1));
            dx  = (x(i+1)-x(i))/ds(i);
            dy  = (y(i+1)-y(i))/ds(i);
            t1  = x(i) - xm1;
            t2  = y(i) - ym1;
            t3  = x(i+1) - xm1;
            t7  = y(i+1) - ym1;
            t4  = t1 * dx + t2 * dy;
            t5  = t3 * dx + t7 * dy;
            t6  = t2 * dx - t1 * dy;
            t1  = t5 * log(t5*t5+t6*t6) - t4 * log(t4*t4+t6*t6);
            t2  = atan2(t6,t4)-atan2(t6,t5);
            a(j,i) = (0.5 * t1-t5+t4+t6*t2)/(2.*pi);
        end
    end
    a(n+1,1) = 1.0;
    a(n+1,n) = 1.0;
end

% Assemble the Right hand Side of the Matrix system

rhs=zeros(n+1,1);
alpha = alpha * pi /180;
xmid=zeros(n,1);

for i = 1:n
  xmid(i,1) = 0.5 * (x(i) + x(i+1));
  ymid = 0.5 * (y(i) + y(i+1));
  rhs(i,1) = ymid * cos(alpha) - xmid(i) * sin(alpha);
end

gamma = zeros(n+1,1);

% Solve the system of equations
% In MATLAB this is easy!

gamma = a\rhs;
cp=zeros(n,1);
cp1=zeros(n,1);

for i = 1:n
    cp(i,1) = 1. - gamma(i) * gamma(i);
    cp1(i,1) = - cp(i,1);
    xa    = xmid(i,1);
```

```
    cpa = cp(i,1);
end

% Compute Lift and Drag Coefficients

cy = 0.0;
cx = 0.0;
cm = 0.0;

% we assume that the airfoil has unit chord
% we assume that the leading edge is at i = nl;

for i=1:n
    dx = x(i+1) - x(i);
    dy = y(i+1) - y(i);

% xarm is the moment arem , equals distance from
% the center of the panel to quarter-chord.

    xarm = 0.5 * (x(i+1)+x(i))-x(nl)-0.25;
    cy = cy - cp(i,1) * dx;
    cx = cx + cp(i,1) * dy;
    cm = cm - cp(i,1) * dx * xarm;
end

% Print Lift and Drag coefficients on the screen

cl = cy * cos(alpha) - cx * sin(alpha);
cd = cy * sin(alpha) + cx * cos(alpha);
cm;
```

process3.m

```
% process3.m
% this is a post-processor for the airfoil design program to save the results

function [x,y,cl,cm,cd,cpc,xmid] = process2(desVar,xu,xL,yu0,yL0,nu,nL,M)

% define alpha
last = length(desVar);
alpha = desVar(last,1);

% compute bump functions
pu = bump3(xu');
pL = bump3(xL');

% compute bump amounts
totVars = length(desVar) - 1;
nVars = totVars/2;
desVaru = zeros(nVars,1);
desVarL = zeros(nVars,1);
desVaru = desVar(1:nVars,1);
desVarL = desVar(nVars+1:totVars,1);
count = nVars - 1;
pertu = zeros(count,nu);
pertL = zeros(count,nL);
for i = 1:count
    pertu(i,:) = pu(i,:).*desVaru(i,1);
    pertL(i,:) = pL(i,:).*desVarL(i,1);
end

dyu = zeros(nu,1);
for j = 1:nu
    dyu(j,1) = sum(pertu(:,j));
end
dyL = zeros(nL,1);
for k = 1:nL
    dyL(k,1) = sum(pertL(:,k));
end

yu = (yu0 + dyu).*desVaru(nVars,1);
yL = (yL0 + dyL).*desVarL(nVars,1);

% stretch y-direction for compressibility correction
corr = xu(nu)./(sqrt(1-M^2));
yu = yu.*corr;
yL = yL.*corr;

% reformat data for panel code
points = nu + nL - 1;
x = zeros(1,points);
y = zeros(1,points);
x(1,nL:points) = xu';
y(1,nL:points) = yu';
for ii = 1:nL
    x(1,nL + 1 - ii) = xL(ii,1);
    y(1,nL + 1 - ii) = yL(ii,1);
end

% run panel code
[cpc,cl,cm,cd,xmid] = panelB(x,y,alpha,nu,nL);
```

seed.m

```
% seed.m
% this file reads in a seed airfoil and formats the data

function [xu,xL,yu,yL,nu,nL] = seed(airfoil)
%function [xun,xLn,yun,yLn,nun,nLn] = seed(airfoil)

% read in airfoil in a tab-delimited file
% format of this file MUST BE:
%   row 1:                  nu   nL
%   row 2 to 1+nu:          xu   yu
%   row nu+1 to nu+nL+1:    xL   yL
% all data must be input from leading edge to trailing edge and must duplicate LE and TE points
if necessary
input = dlmread(airfoil,'\t');

% specify number of upper and lower surface points
nu = input(1,1);
nL = input(1,2);

% read in upper surface points
uend = nu + 1;
xu = input(2:uend,1);
yu = input(2:uend,2);

% read in lower surface points
Lstart = uend + 1;
Lend = Lstart + nL - 1;
xL = input(Lstart:Lend,1);
yL = input(Lstart:Lend,2);
```