The Adjoint Method for Optimal Aerodynamic Design on Unstructured Grids

by

Dan Carney

A thesis submitted in conformity with the requirements
for the degree of Masters of Applied Science
Graduate Department of Aerospace Science and Engineering
University of Toronto

Abstract

The Adjoint Method for Optimal Aerodynamic Design on Unstructured Grids

Dan Carney
Masters of Applied Science
Graduate Department of Aerospace Science and Engineering
University of Toronto
2003

One of the fastest methods for automated aerodynamic shape optimization is available only on structured grids. It employs a Krylov-subspace solver to solve a set of discrete adjoint equations in $\frac{1}{5}$ to $\frac{1}{2}$ the time of a Newton-Krylov flow solve. With the solution to these equations, the optimization method uses the BFGS Hessian-approximation formula and a backtracking line search to efficiently solve many optimization problems based on the Navier-Stokes equations. An extension of this method to unstructured grids for the Euler equations is presented. Aside from the difference in spatial discretization and flow equations, the major difference in the two methods is the grid-movement algorithm, for which an efficient, modified version of a common algorithm is presented. The unstructured optimization method is proven over a number of cases that include drag minimization and inverse design, and it is found to be as efficient as the structured method.

Contents

List of Tables

List of Figures

List of Symbols and Abbreviations

Calligraphic Symbols

$\mathcal{D}$	Dissipation Contribution
$\mathcal{F}$	Flow Equations
$\mathcal{G}$	Objective-Function Derivative
$\mathcal{H}$	Approximate Hessian Inverse
$\mathcal{J}$	Objective Function
$\mathcal{L}$	Biharmonic Operator
$\mathcal{X}$	Parameter Set

Alphanumeric Symbols

a	Sound Speed
B	B-Spline Basis Function
C	B-Spline Point
d	Knot Vector
e	Energy
F	Inviscid Flux
H	Stagnation Enthalpy
h	Finite Step Size
I	Identity Matrix
i	Dummy Variable

j	Dummy Variable
K	B-Spline Order
L	Distance Between Surface Points
l	Control Polygon Length
M	Flow Mach Number
m	Penalty Function Target
N	Number of Faces
n	Surface Normal
P	Grid Point
p	Pressure
Q	Conserved Flow Variables
R_1, R_2, R_3, R_4	Riemann Invariants
s	Search Direction
t	Time
u	x Component of Flow Velocity
V	Control Volume
v	y Component of Flow Velocity
w	Spline Parameters
X	B-Spline Control Point
x	First Basis of Cartesian Space
Y	B-Spline Control Point
y	Second Basis of Cartesian Space
C_D^*	Target Drag Coefficient
C_D	Drag Coefficient
C_L^*	Target Lift Coefficient

C_L Lift Coefficient

C_M^* Target Moment

C_M Quarter Chord Moment

C_P^* Target Pressure Coefficient

C_P Pressure Coefficient

V_n Normal Velocity

V_t Tangential Velocity

Greek Symbols

β Step Size

ϵ Basis of Parameter Space

η B-Spline Dimension

γ Gamma

λ Dissipation Scale

ν Objective-Function Derivative Difference

Ω Parameter Space

ϕ Adjoint

ϕ Line Search Function

ρ Density

τ Parameters Difference

$\varepsilon^{(2)}$ Second-Order Dissipation Scale

$\varepsilon^{(4)}$ Fourth-Order Dissipation Scale

$\kappa^{(2)}$ Second-Order Dissipation Coefficient

$\kappa^{(4)}$ Fourth-Order Dissipation Coefficient

ω_D Weight Coefficient Related to Drag Coefficient

ω_L Weight Coefficient Related to Lift Coefficient

ω_M Weight Coefficient Related to Moment

ω_P Weight Coefficient Related to Penalty Function

Abbreviations

BFGS Broyden-Fletcher-Goldfarb-Shanno

GMRES Generalized Minimum Residual

ILU Incomplete Lower Upper

RCM Reverse Cuthill-Mckee

Chapter 1

Introduction

1.1 Motivation

A human designing an aircraft encounters numerous unknowns and tradeoffs, as an aircraft must be designed with a range of takeoff, landing, and flight conditions. By generating a good aerodynamic shape for the aircraft, the designer significantly affects all of these conditions. However, a good aerodynamic shape may induce problems such as too high a weight, too small a fuel tank, too weak an internal structure, or too expensive an aircraft to build. The designer must take all of these things into consideration.

That being said, the designer is faced with one problem: variables. There are too many. Because the equations with which the designer works are all nonlinear, it is completely unclear what even a small change in one variable or a set or variables will do to the resulting aircraft. If altering the wingtip in one way has some effect, then further altering the wingtip in the same manner could have exactly the opposite effect. Traditionally, searching the design space was a difficult task in itself. If a windtunnel was used, multiple wingtip-mockups would have to be built, just to find one that performs well. Moreover, supposing the designer finds the optimal such wingtip for a particular set of variables, that wingtip may no longer be at all optimal if the thickness of the wing is changed slightly at the wing-body interface. The designer must optimize everything at once, a humanly-impossible task. The traditional "cut and try" approach to aircraft design, especially one using windtunnels, is highly inefficient.

Luckily, the designer is not hopelessly lost. Today, a computer can perform the task of finding an optimal design if it is given a design space and a definition of what is optimal. Thus, the designer's task is to define these for the computer, a complex task but one which the computer is not capable of doing itself.

1.2 Goal

On structured grids, an optimization method has been developed that is extremely fast and efficient [19, 22, 20, 21, 23]. This method is, however, prone to the same problems that structured grids always have: poor resolution in some blocks, greater resolution than necessary in other blocks, and problems conforming to complex shapes. Also, the grid can be useless for computation if it is deformed too much [18]. The goal of the work in this paper is to extend the structured method to an unstructured method that is equally efficient. Because the flow equations are usually more highly coupled in unstructured grids, the unstructured version will be slightly slower than the structured one. The goal is to have a method with the same computational order.

1.3 Overview of Method

1.3.1 Flow Equations

The first task of the designer is to decide upon a discretized set of flow equations and a method for solving them. Because the flow calculated on one grid will not necessarily be the same as the flow calculated on another, an optimal design on one grid will not necessarily be optimal on another. Moreover, an optimal design using one set of flow equations will not necessarily be optimal using another set of flow equations. The designer must choose a configuration that is known to have an acceptable solution.

In this work, the 2D Euler equations are solved. Additionally, artificial dissipation is added for numerical stability. This artificial dissipation is an unstructured version of the well known Jameson, Schmidt, Turkel equations [12]. The outer-boundary conditions are based on Riemann invariants, and the inner on a "zero tangential flow" condition. The grid itself is arbitrarily unstructured, and, in general, a grid of primitive shapes is first constructed, and the flow is solved on the centroidal-median dual of that primitive grid. The flow solver employed is an efficient Newton-Krylov solver [31, 14]. Good optimization results have been already obtained using the structured-grid method for Euler, laminar, and turbulent flows [18]. Thus, verifying the new method on the Euler equations is an important first step.

1.3.2 Parameterization

The designer must be parameterize the problem at hand. In particular, a finite set of parameters in a space of reasonable parameters is used to describe the possible shapes an airfoil may take. This parameter space may have any number of constraints. For instance, it may be necessary to constrain the thickness of the airfoil so that the resulting aircraft is not structurally useless

or too heavy to fly. For whole aircraft configurations, the designer must find a more complex parameterization, one that represents enough variety of aircraft to contain a sufficiently optimal one but also constrains the design space to aircraft that can be reasonably manufactured.

Shenoy and Heinkenschloss [30] as well as Giles and Pierce [9] use simple algebraic functions to parameterize an airfoil surface. Samareh [29] gives a good overview of possible shape parameterizations. Nemec [18], on whose work this is based, has shown that an airfoil may be represented as a spline, and the parameters can be a subset of the spline control-points [5].

1.3.3 Grid Deformation

One issue that arises during parameterization is grid deformation. Most, if not all the parameters used in an optimization will deform the aerodynamic shape. This implies that the computational grid must deform as well. Moreover, this deformation may have to have differentiablity properties, depending on the type of optimization method used. A number of methods exist for deforming the grid, the most common of which makes the analogy that the computational grid is a vast web of springs in which the cell interfaces are the springs and the nodes are the spring endpoints.

Nemec [18] uses a simple perturbation formula suitable only on structured grids. Variations on the spring analogy on unstructured grids are used in the work of many researchers, although none of them use an algorithm with differentiability properties [24, 6, 7, 9]. However, Nielson and Anderson use an isotropic linear elasticity analogy for improved, though not perfect, differentiablity on a triangular grid [25].

In the present work, the segment-spring analogy is used with a boundary correction. This means that each spring has an equilibrium length and a stiffness associated with it, and that the springs are stiffer near the computational boundaries. This method has been shown to work well with viscous-grid deformation, and is thus suitable not only for the Euler equations but the extension of the method to laminar and viscous flows. Also, most researchers use a slow technique, Jacobi iterations, to solve the linear system of the spring analogy. An efficient Krylov-subspace method for solving this system is presented.

1.3.4 Objective Function

The designer must come up with a real-valued function of the parameters that represents the quantity to be optimized [11]. This function, the objective function, is generally a function for which optima are local minima. The ratio of drag to lift is a simple example of such a function, although it need not be simple. Constraints may also be added here in the form of penalty functions. That is, if an airfoil tends towards a bad shape, the objective function is increased.

In this manner, constraints can be softer; a set of parameters does not have to be either in the constrained space or outside of it. The present work uses a variety of simple objective functions that are designed for good verification of the optimization method.

1.3.5 Optimization

There are a number of common methods for optimizing the objective function. For example, genetic algorithms can find a global optimum using ideas based on natural selection. Other methods are only capable of finding local optima. In particular, derivative-based methods [11] can, in few flow solves, generate a good approximation of the local curvature of the design space. This requires that the design space be continuous and differentiable. Many researchers have found derivative based-methods to converge well [16, 24, 6]. Genetic algorithms, although they do not require differentiability, require many flow solves to converge to the global minimum. Goldberg [10] and Coello [3] both provide good overviews of these techniques.

Some work has also been done on fully-coupled methods or "all at once" design. With these methods an optimum, not necessarily global, is found in one nonlinear solve. The derivative-based methods are actually an uncoupled form of the same nonlinear system. Shenoy et al. [30] and Gatsis and Zingg [8] have applied fully-couped methods to aerodynamic problems and found that they require only 3 to 6 times the cost of an analysis problem but are prone to poor convergence.

One reason for studying the derivative-based methods is that they are, unlike fully-coupled ones, robust. Also, the designer can thus narrow the parameter space so that the deformations in the shapes are small, and this makes it more likely that a design space has one local minimum, making the global and local methods converge to the same location. The same is true of a design space with a finite number of discontinous regions. For example, if a parameterization decides the number of nacelles on a wing, the objective function is most likely discontinous every time it moves from 1 to 2 nacelles. However, local optimization applied twice to the problem, once for 1 nacelle and once for 2 nacelles, should converge to the global optimum if the rest of the design space is considerably small. Thus, for small design spaces, the derivative-based methods are currently more robust than fully-coupled methods and more efficient than global methods. These are not trivial observations, as aircraft are usually designed to be modifications of previous aircraft. Also, designers generally face economic concerns and slightly modifying an aircraft makes it cheaper to build and test.

The present work is an extension of a derivative-based method. However, it differs from similar methods on unstructured grids. For instance, because the grid is differentiable, more accurate derivatives can be computed, which Neilson and Anderson have concluded to be worthwhile for robustness [25]. Also, the Newton-Krylov preconditioner that makes the flow solver

efficient makes the derivative calculation efficient as well, and except the in work of Nemec, this preconditioner is unused [24, 7, 9, 21].

1.3.6 Derivative-Based Methods

Derivative-based methods, often called gradient-based methods, are so called as they rely on the ability to calculate a first derivative, or gradient, of the objective function with respect to each design variable. These methods are built of two distinct parts: the calculation of the derivative, and the use of this derivative to find a better approximation of the local minimum.

There are four methods currently available for calculating the derivative: the finite-difference method, the complex-step method, the sensitivity method, and the adjoint method. The sensitivity method and the adjoint method are further subdivided into discrete and continuous approaches. In the discrete case, the equations are discretized and then differentiated, and in the continuous case, the equations are differentiated and then discretized. The computational costs are similar for each, and a number of studies have shown the discrete form to be slightly more robust due to more accurate derivatives [16]. The three methods can be compared on the basis of the most computationally expensive components [17, 26, 15]. Table 1.1 shows that the adjoint method is the most efficient for large systems, and this is the method used in the present optimizer.

With this derivative information, the simplest action is to move the parameters a step in the opposite direction of the derivative. This method is prone to failure on most spaces. However, Nadarajah and Jameson[16] have had success with a smoothed derivative and a small step in the direction of steepest descent. There are numerous other ways to generate a sequence of steps for optimization. Often, the Kreisselmeier-Steinhauser function is used [32]. A number of methods have been developed to approximate the curvature of the objective function given a series of derivatives. An overview of these methods is given by Gunzburger [11]. The most commonly used such method is the Broyden-Fletcher-Goldfarb-Shanno (BFGS) method. It is considered the most accurate method for generating an approximation to the inverse of the Hessian of the objective function [26, 33].

Method	**Order of Computational Cost**
finite difference	one system solve for every parameter
complex step	one system solve for every parameter
sensitivity	one linear solve for every parameter
adjoint	one linear solve

Table 1.1: A Comparison of Derivative Methods

Chapter 2

Equations and Algorithms

2.1 Flow Equations

The the two-dimensional Euler equations over a volume, V, is

$$\frac{d}{dt}\int_V Q\,dv + \oint_{\partial V} F \cdot n\,da = 0 \tag{2.1}$$

where

$$Q = \begin{bmatrix} \rho \\ \rho u \\ \rho v \\ e \end{bmatrix} \tag{2.2}$$

and

$$F = \begin{bmatrix} \rho u, \rho v \\ \rho u^2 + p, \rho uv \\ \rho uv, \rho v^2 + p \\ u(e+p), v(e+p) \end{bmatrix} \tag{2.3}$$

These equations are taken to be thermally perfect, so

$$p = (\gamma - 1)\left[e - \frac{1}{2}\rho(u^2 + v^2)\right] \tag{2.4}$$

2.1.1 Discretized Flux

For cell i, with volume V_i and N_i number of faces, the flow is solved by assuming that the cell average Q_i is constant along each cell edge. With n_{ij} defined as the normal of cell i's j^{th} edge, Eq. 2.1 becomes

$$V_i \frac{d}{dt} Q_i + \sum_{j=1}^{N_i} F_i \cdot n_{ij} = 0 \tag{2.5}$$

However, to force the flux across an interface to be identical for both cells, an average flux is used.

$$\frac{1}{2}(F_i + F_j) \cdot n_{ij} \tag{2.6}$$

2.1.2 Artificial Dissipation

To introduce numerical stability, an artificial dissipation term is added to the flux term across each face in the domain [12]. There is no dissipation applied to boundary faces. This dissipation scheme has been proven to stabilize the solution in many flow regimes. It consists of a first-order operator designed to capture shocks crisply, and a second-order operator which provides greater stability. The degree to which these operators are applied is controlled by a pressure switch. The operator applied to the cell i's j^{th} edge is

$$\mathcal{D}_{ij} = \frac{1}{2}\lambda_{ij}\left[\frac{\varepsilon_i^{(2)} + \varepsilon_j^{(2)}}{2}(Q_j - Q_i) - \frac{\varepsilon_i^{(4)} + \varepsilon_j^{(4)}}{2}(\mathcal{L}_j - \mathcal{L}_i)\right] \tag{2.7}$$

where the dissipation scaling is

$$\lambda_{ij} = |(u_{ij}, v_{ij}) \cdot n_{ij}| + a_{ij}|n_{ij}| \tag{2.8}$$

The second-difference operator scaling, the pressure switch, is

$$\varepsilon_i^{(2)} = \kappa^{(2)} \frac{\sum_{j=1}^{N_i} |p_j - p_i|}{\sum_{j=1}^{N_i} p_j + p_i} \tag{2.9}$$

The fourth-difference operator scaling is

$$\varepsilon_i^{(4)} = \max(0, \kappa^{(4)} - \varepsilon_i^{(2)}) \tag{2.10}$$

and

$$\mathcal{L}_i = \sum_{j=1}^{N_i} (Q_j - Q_i) \tag{2.11}$$

2.1.3 Boundary Conditions

Boundaries are extrapolated based on a zeroth-order Taylor expansion:

$$Q_{e_i} = Q_i \tag{2.12}$$

Body Boundary

At the body boundary we assume that the normal flow velocity is zero.

$$V_n = (u, v) \cdot n = 0 \tag{2.13}$$

Where n is the outward normal of the boundary face. Stagnation enthalpy is defined as

$$H = (e + p)/\rho \tag{2.14}$$

and is set to the free-stream value. The tangential velocity is

$$V_t = (v, -u) \cdot n \tag{2.15}$$

and the density at the interface is

$$\rho_b = \frac{\gamma p_e}{(\gamma - 1)(H_\infty - \frac{1}{2}{V_{te}}^2)} \tag{2.16}$$

The flow velocities are defined as

$$u_b = (V_{ne}, -V_{te}) \cdot n \tag{2.17}$$

and

$$v_b = (V_{te}, V_{ne}) \cdot n \tag{2.18}$$

and the energy is

$$e_b = \frac{p_e}{\gamma - 1} + \frac{1}{2}\left[\frac{\gamma p_e}{(\gamma - 1)(H_\infty - \frac{1}{2}{V_{te}}^2)}\right]{V_{te}}^2 \tag{2.19}$$

Far-Field Boundary

Nonreflecting boundary-conditions are applied at the far-field boundary. One-dimensional Riemann invariants are used.

$$R_1 = V_n - \frac{2a}{\gamma - 1} \tag{2.20}$$

$$R_2 = V_n + \frac{2a}{\gamma - 1} \tag{2.21}$$

$$R_3 = \frac{p}{\rho^\gamma} \tag{2.22}$$

$$R_4 = V_t \tag{2.23}$$

Various boundary values are calculated from these invariants.

$$V_{nb} = \frac{1}{2}(R_1 + R_2) \tag{2.24}$$

$$a_b = \frac{1}{4}(\gamma - 1)(R_1 - R_2) \tag{2.25}$$

$$\rho_b = \left(\frac{a_b^2}{\gamma R_3}\right)^{\frac{1}{\gamma-1}} \tag{2.26}$$

$$u_b = (V_{nb}, -R_4) \cdot n \tag{2.27}$$

$$v_b = (R_4, V_{nb}) \cdot n \tag{2.28}$$

$$e_b = \frac{\rho_b a_b^2}{\gamma(\gamma - 1)} + \frac{1}{2}(u_b^2 + v_b^2) \tag{2.29}$$

The choice of which Riemann invariants to use depends on the local flow velocity and is summarized in Table 2.1. Although no supersonic cases will be presented in this work, transonic cases will, and these often have supersonic regions at the boundary when the flow solver is starting up.

	Inflow				Outflow			
invariant	R_1	R_2	R_3	R_4	R_1	R_2	R_3	R_4
subsonic	Q_∞	Q_e	Q_∞	Q_∞	Q_∞	Q_e	Q_e	Q_e
supersonic	Q_∞	Q_∞	Q_∞	Q_∞	Q_e	Q_e	Q_e	Q_e

Table 2.1: Riemann Invariants

2.2 Flow Solution Algorithm

The flow is solved using an efficient Newton-Krylov method [14]. At each iteration, the method attempts to reduce the norm of $\mathcal{F}$, where

$$\mathcal{F}_i = \sum_{j=1}^{N_i} \frac{1}{2}(F_i + F_j) \cdot n_{ij} + \mathcal{D}_{ij} + \sum_{j=1}^{N_{e_i}} F_{e_j} \cdot n_{ie_j} \tag{2.30}$$

It does so by generating a first-order approximation of $\mathcal{F}^{(i+1)}$.

$$\left(\frac{\partial \mathcal{F}}{\partial Q}\right)^{(i)} \Delta Q^{(i)} = -\mathcal{F}^{(i)} \tag{2.31}$$

This is a linear system which is solved using the Generalized Minimal Residual algorithm GMRES(35) [28]. The system is preconditioned using incomplete lower-upper (ILU) factorization. ILU(5) or ILU(7) is used depending on the difficulty of the case. The preconditioner upon which the ILU factorization is based has been proven to be very efficient for this system [31]. Also, the cells are reordered according to the Reverse Cuthill-Mckee algorithm (RCM) [4] for

further speedup. Once the residual of the linear system is reduced in magnitude by 0.1, the system is set up with:

$$Q^{(i+1)} = Q^{(i)} + \Delta Q^{(i)} \tag{2.32}$$

Startup is handled by adding a local timestep to the left-hand side of the linear system, creating an implicit Euler algorithm that is necessary in transonic cases. These iterations are continued until machine zero is reached.

2.3 Parameterization

A simple parameterization for an airfoil is based on an K-order spline [5, 18] As the order of the spline increases, the control points become coupled to more surface points and the surface becomes more smooth. In the present work, the splines are all order 3. The spline approximates η points on an airfoil surface given by an ordered set of consecutive edge points, P. The spline is a sum of basis functions B:

$$B_{j,1}(w_i) = \begin{cases} 1 & d_j \leq w_i \ < d_{j+1} \\ 0 & \text{otherwise} \end{cases} \tag{2.33}$$

$$B_{j,K}(w_i) = \frac{w_i - d_j}{d_{j+K-1} - d_j} B_{j,K-1}(w_i) + \frac{d_{j+K} - w_i}{d_{j+K} - d_{j+1}} B_{j+1,K-1}(w_i) \tag{2.34}$$

Here, d is the set of knots, w is a set of parameters.

The knot vector is based on the shape of an airfoil and is used to cluster control points where the most significant curvature is. The following knot vector forces the spline endpoints to coincide with the first and last control point. Also, the control points are clustered near the middle of the spline.

$$d_j = \begin{cases} 0 & 1 \leq j \leq K \\ \frac{\eta - K + 2}{2} \left[1 - \cos\left(\frac{j-K}{\eta - K + 2} \pi \right) \right] & K+1 \leq j \leq \eta + 1 \\ \eta - K + 2 & \eta + 2 \leq j \leq \eta + K + 1 \end{cases} \tag{2.35}$$

This knot vector is useful for simple airfoils since the more important leading-section of the airfoil can be located in the middle of the spline, getting more control points, and the less important trailing-section will get fewer control points. The spline parameters are chosen according to the distances between consecutive grid nodes, L:

$$w_i = \begin{cases} 0 & i = 1 \\ \frac{\eta - K + 2}{\sum_{j=1}^{\eta - 1} \sqrt{L_j}} \sum_{j=1}^{i-1} \sqrt{L_j} & \text{otherwise} \end{cases} \tag{2.36}$$

The control points are chosen to minimize the square of the distances between the points on the airfoil and the corresponding points on the spline. The coordinates of these points are

$$C_i = \left[\sum_{j=1}^{\eta+1} X_j B_{j,K}(w_i), \sum_{j=1}^{\eta+1} Y_j B_{j,K}(w_i) \right] \tag{2.37}$$

An iterative refinement brings the spline points closer to the edge points. It does this by first modifying the spline parameters. w_i is increased by

$$(P_i - C_i) \frac{C_i'}{\|C_i'\|} \frac{d_{\eta+1} - d_1}{l} \tag{2.38}$$

where l is the sum of the distances between consecutive control points. Then, the control points are solved for again. This process is repeated until the maximum difference between the spline points and the edge points is less than 5×10^{-4} chords.

2.4 Grid Deformation

It is important that the location of each grid node is a differentiable function of the parameters. If it is not, the objective function will not be a differentiable function of the parameters, and optimization will fail. Quite commonly, grid deformation algorithms do not consider differentiability, and conforming the grid to one set of parameters and back to the original set generates quite a different grid. In this case, the grid is not even a function of the parameters, let alone a differentiable one.

The segment-spring algorithm is a common grid-movement algorithm that can be made differentiable. All grid nodes are a linear, thus differentiable, function of the boundary nodes, which in turn are a differentiable function of spline control-points. When combined with a boundary improvement, it can perform fairly large deformations on grids suitable for viscous flows [1].

In this method, a force function on every point in the domain is generated when the points on the boundary move. The force at some point i is

$$F_i = \sum_{j=1}^{N_i} \alpha_{ij} (\delta_j - \delta_i) \tag{2.39}$$

where N_i is the number of lattice neighbours of i, and δ_i is the vector displacement of point i from equilibrium. α_{ij} is a stiffness coefficient that determines the stiffness of the lattice between point i and point j. To ensure differentiability of the grid, the stiffness coefficients are computed only once, and each computation is performed assuming that the displacement computed is a displacement from the original node location. Normally, this is not the case. Instead, several

small interpolated steps are used with updated coefficients. This violates the need for the grid to be a function of the parameters.

The values of the stiffness coefficients are important for maintaining the boundary mesh. A useful formula for these coefficients is

$$\alpha_{ij} = \phi[(x_i - x_j)^2 + (y_i - y_j)^2]^{\Psi} \tag{2.40}$$

As mentioned before, for differentiability, these coefficients are based on the initial values of the grid nodes. For the outer region, good values of ϕ and Ψ are $\phi = 1$ and $\Psi = -1$ or $\Psi = -\frac{1}{2}$. Near the boundaries, good values are $\phi = 5$ and $\Psi = -1$ [2, 1].

The static equilibrium of the system is reached when $F_i = 0$. This set of equations is a sparse linear system, which, traditionally, is done using Jacobi iterations. This means iterating

$$\delta_i^{k+1} = \frac{\sum_{j=1}^{N_i} \alpha_{ij} \delta_j^k}{\sum_{j=1}^{N_i} \alpha_{ij}} \tag{2.41}$$

over all points in the domain until δ_i^{k+1} is sufficiently close to δ_i^k [2]. Then, the new point positions are determined by adding the displacements to the original points. This process is accurate and consistent but slow.

A much faster method of solving the linear system is to use one of a multitude of sparse linear-solvers. Essentially the same method used for the flow solves works well for this. That is, the system is right preconditioned and solved efficiently via GMRES(30). The preconditioner is an ILU(2) decomposition based on the left-hand side of the system. This preconditioner is necessary for fast convergence of the solve and only needs to be computed once, as all successive solves require the same preconditioner and left-hand side. Smaller values of fill also work, but 2 was found to be ideal for the grids in the present work. RCM reordering also contributes a small speedup to the algorithm and is thus used since the time of computing the reordering is only spent once. The system has the size of the number of grid points used. Since this number can be quite large, the linear system is solved to machine zero. This ensures that the grid movement algorithm will be as close to deterministic as possible for a given parameterization. The equation for an interior point i is just Eq. 2.39 with $F_i = 0$. For a boundary point, the diagonal entry and the corresponding right-hand side entry are both 1. This method converges orders of magnitude faster than the Jacobi method in all problem sizes. For example, on a grid of 3000 nodes, the new method is more than 500 times faster.

It is feasible and advantageous to solve the linear system on only a collection of points near the airfoil boundaries and leave the remaining points in the domain fixed. To do this, the inner boundary faces are all collected. Then all cells with points touching these faces are collected, forming a layer of cells around the airfoil. This process is repeated with the boundary

edges of the cell layer, building up layers, until enough layers have been collected to effectively move the airfoil through a full range of reasonable movement. The six innermost cell layers are computed with the stiff set of coefficients, as are the outermost six layers. Fifteen layers of cells between these layers are computed with the less stiff coefficients, forcing most of the deformation to occur in the these layers. This method, because it reduces the size of the linear system, increases the accuracy with which the points are found, thus increasing deterministic calculation and differentiability. Figs. 2.1 and 2.2 show some results of the grid deformation algorithm. The NACA 0012 airfoil in Fig. 2.1 was highly deformed, and the grid around it was moved to accommodate.

2.5 Objective Functions

An objective function, $\mathcal{J}$, of design variables, $\mathcal{X}$, and flow variables, Q, will be considered. The goal of optimization will be to find

$$\min_{\mathcal{X}\in\,\Omega} \mathcal{J}[\mathcal{X}, Q(\mathcal{X})] \tag{2.42}$$

2.5.1 Inverse Design

The inverse design objective function has an optimal set of parameters when the pressure distribution around an airfoil is the same as a target pressure-distribution [22]. Since it is unlikely that two airfoils will have the same pressure distribution at the same flow conditions, this function can usually find the unique airfoil described by the target pressure distribution. Thus, if a pressure distribution around a known airfoil is used as the target pressure distribution, this particular objective function can verify the optimizer by forcing the objective function to machine zero. This function is

$$\mathcal{J}[\mathcal{X}, Q(\mathcal{X})] = \sum_{i=0}^{N} \{C^*_{Pi} - C_{Pi}[\mathcal{X}, Q(\mathcal{X})]\}^2 \tag{2.43}$$

where N is the number of airfoil boundary edges.

2.5.2 Target Coefficients

Another important objective function, more useful for simple aerodynamic-design, is one that targets particular lift and drag coefficients [22]:

$$\begin{aligned} \mathcal{J}[\mathcal{X}, Q(\mathcal{X})] &= \omega_L \{C_L[\mathcal{X}, Q(\mathcal{X})] - C^*_L\}^2 + \\ &\quad \omega_D \{C_D[\mathcal{X}, Q(\mathcal{X})] - C^*_D\}^2 + \omega_M \{C_M[\mathcal{X}, Q(\mathcal{X})] - C^*_M\}^2 \end{aligned} \tag{2.44}$$

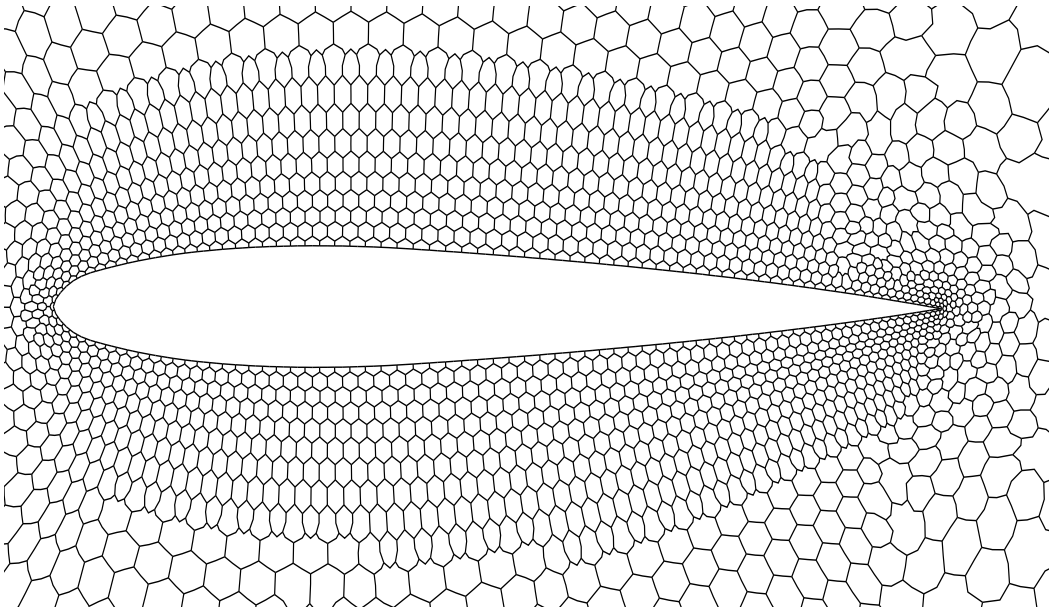

Figure 2.1: Before Grid Movement

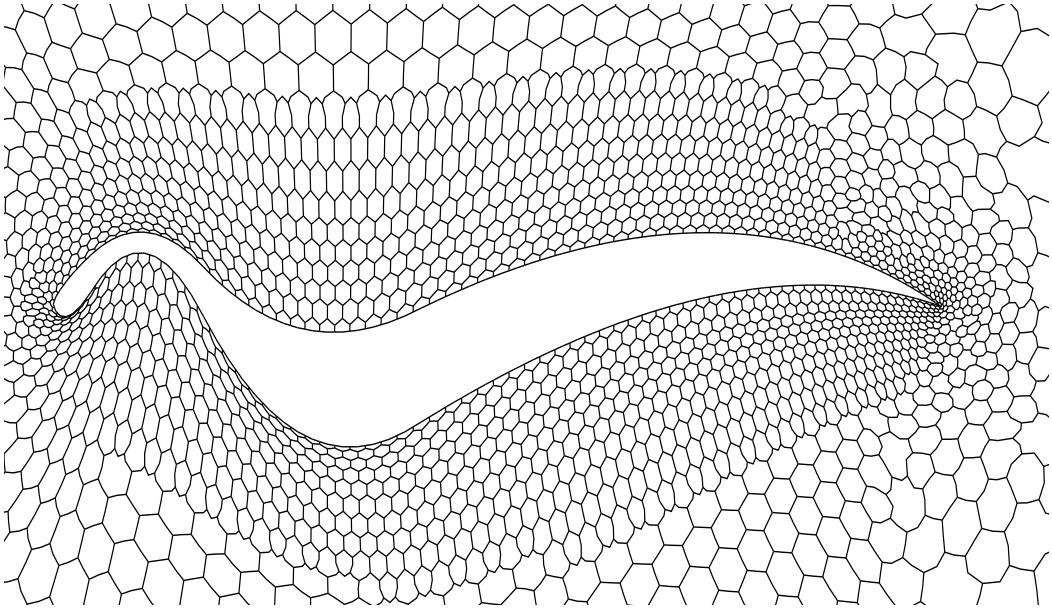

Figure 2.2: After Grid Movement

where ω_L, ω_D and ω_M are fixed weights, and C_L^*, C_D^*, and C_M^* are the target lift coefficient, drag coefficient and quarter-chord moment.

2.6 Penalty Function

Where needed a simple penalty function is applied to keep the airfoil geometry within reasonable values. A simple, but effective penalty function increases the penalty quadratically when two grid points, p_1 and p_2, come closer than a minimum value, m. A sum of these functions can be written as:

$$\mathcal{J}(\mathcal{X}) = \sum_i \begin{cases} 0 & |p_{1i}(\mathcal{X}) - p_{2i}(\mathcal{X})| - m_i > 0.0 \\ \omega_{Pi}(|p_{1i}(\mathcal{X}) - p_{2i}(\mathcal{X})| - m_i)^2 & \text{otherwise} \end{cases} \tag{2.45}$$

where ω_P is a weight assigned to the function.

2.7 Derivatives

In this paper, the most computationally-efficient method, the adjoint method, will be used, but the finite-difference method will be used for accuracy comparisons.

2.7.1 Finite-Difference Derivative

The finite-difference derivative is computed using the following formula:

$$\left(\frac{d\mathcal{J}}{d\mathcal{X}}\right)_i = \frac{\mathcal{J}\left[\mathcal{X} + h\epsilon_i, Q(\mathcal{X} + h\epsilon_i)\right] - \mathcal{J}\left[\mathcal{X} - h\epsilon_i, Q(\mathcal{X} - h\epsilon_i)\right]}{2h} \tag{2.46}$$

where i ranges over the number of parameters in the parameter space, e_i is the i^{th} basis of the parameter space, and h is a small value related to the scaling of the parameters. In order to get accurate results, h can neither be too large or too small. In the case of spline control point parameters around a NACA 0012 airfoil scaled to unity, a good h is between 10^{-4} and 10^{-5}. For second-order accuracy, a centered-difference formula is used. This requires two flow solves for every parameter. This is the most straightforward method of calculating the derivative, as it can be programmed on top of the flow solver without any knowledge of flow solver internals.

2.7.2 Adjoint Derivative

More information is available than that used by the finite-difference method. The objective function contains information about the flow around the airfoil, as modelled by the Euler equations, which can be represented as

$$\mathcal{F}[\mathcal{X}, Q(\mathcal{X})] = 0 \qquad \forall \mathcal{X} \in \Omega \tag{2.47}$$

This implies that

$$\frac{d\mathcal{F}}{d\mathcal{X}} = 0 \tag{2.48}$$

The derivative, then, of $\mathcal{J}$ is given by

$$\frac{d\mathcal{J}}{d\mathcal{X}} = \frac{\partial \mathcal{J}}{\partial \mathcal{X}} + \frac{\partial \mathcal{J}}{\partial Q}\frac{dQ}{d\mathcal{X}} \tag{2.49}$$

Similarly, the derivative of $\mathcal{F}$ is

$$\frac{d\mathcal{F}}{d\mathcal{X}} = \frac{\partial \mathcal{F}}{\partial \mathcal{X}} + \frac{\partial \mathcal{F}}{\partial Q}\frac{dQ}{d\mathcal{X}} \tag{2.50}$$

which implies, in light of Eq. 2.48,

$$-\frac{\partial \mathcal{F}}{\partial \mathcal{X}} = \frac{\partial \mathcal{F}}{\partial Q}\frac{dQ}{d\mathcal{X}} \tag{2.51}$$

which means that we can rewrite Eq. 2.49 as

$$\frac{d\mathcal{J}}{d\mathcal{X}} = \frac{\partial \mathcal{J}}{\partial \mathcal{X}} - \frac{\partial \mathcal{J}}{\partial Q}\left(\frac{\partial \mathcal{F}}{\partial Q}\right)^{-1}\frac{\partial \mathcal{F}}{\partial \mathcal{X}} \tag{2.52}$$

The partial derivatives w.r.t. $\mathcal{X}$ in Eq. 2.49 can be calculated using the same second-order centered finite-difference formula that was used to calculate the finite-difference derivative. That is

$$\left(\frac{\partial \mathcal{J}}{\partial \mathcal{X}}\right)_i = \frac{\mathcal{J}[\mathcal{X} + h\epsilon_i, Q(\mathcal{X})] - \mathcal{J}[\mathcal{X} - h\epsilon_i, Q(\mathcal{X})]}{2h} \tag{2.53}$$

and

$$\left(\frac{\partial \mathcal{F}}{\partial \mathcal{X}}\right)_i = \frac{\mathcal{F}[\mathcal{X} + h\epsilon_i, Q(\mathcal{X})] - \mathcal{F}[\mathcal{X} - h\epsilon_i, Q(\mathcal{X})]}{2h} \tag{2.54}$$

These computations can be efficiently computed because no flow solves are required. In fact, it is generally possible to compute them exactly without changing the parameters at all, but that requires a more detailed knowledge of the objective function, which may be hard to compute. Now, solving the linear system

$$\frac{\partial \mathcal{F}}{\partial Q}^T \phi = \frac{\partial \mathcal{J}}{\partial Q}^T \tag{2.55}$$

where ϕ is the adjoint, the final form of Eq. 2.49 is

$$\begin{aligned}\left(\frac{d\mathcal{J}}{d\mathcal{X}}\right)_i &= \frac{\mathcal{J}[\mathcal{X} + h\epsilon_i, Q(\mathcal{X})] - \mathcal{J}[\mathcal{X} - h\epsilon_i, Q(\mathcal{X})]}{2h} \\ &\quad -\phi^T \frac{\mathcal{F}[\mathcal{X} + h\epsilon_i, Q(\mathcal{X})] - \mathcal{F}[\mathcal{X} - h\epsilon_i, Q(\mathcal{X})]}{2h}\end{aligned} \tag{2.56}$$

The major computational cost of this derivative evaluation is contained in the solution of the linear system in Eq. 2.55. The left-hand side of this system is the transpose of the system in the flow solver and thus can be solved the same way. That is, a GMRES(35) solver and RCM node reordering is used. The transpose of the matrix upon which the Newton-Krylov preconditioner is based is used to calculate an ILU(7) preconditioner for the solver. The computional costs of the linear solve are about equal to those of the Newton-Krylov method when the flow is

converged. However, the Newton-Krylov method requires convergence only to one order of magnitude at each step whereas the adjoint is converged to machine zero, so a single solve takes slightly longer. In the structured case the computational cost of solving this system is generally $\frac{1}{2}$ to $\frac{1}{3}$ cost of a full flow solve [22], and the same is true of the unstructured method.

2.8 Optimizer

We solve an unconstrained minimization problem, but the objective function can include a penalty function, and regions of the parameter space can be excluded from being searched. If we design this space correctly, we can effectively solve a constrained problem. We use the BFGS quasi-Newton algorithm coupled with a backtracking line-search. The BFGS algorithm is proven to converge for a number of function classes, and although the objective functions are not necessarily in those classes, the algorithm works on them as well. The backtracking search has been effective in reducing CPU time for a number of problems compared with an exact line-search.

The goal of the algorithm is to generate, over a number of steps, $\mathcal{H}$, an approximation to the inverse Hessian of the objective function, and use this approximation along with the derivative, $\mathcal{G}$, to create a search direction, s:

$$s_i = -\mathcal{H}_i \mathcal{G}_i \tag{2.57}$$

With the search direction computed, the line search generates a step size, β. Then the parameter set is updated via

$$\mathcal{X}_{i+1} = \mathcal{X}_i + \beta s_i \tag{2.58}$$

When the inverse Hessian is exact and $\beta = 1$, this step corresponds to the search direction derived from a second-order Newton's method.

The updated parameters are required to satisfy the strong Wolfe conditions [13], which ensure the convergence of the inverse Hessian:

$$\mathcal{J}[\mathcal{X}_i + \beta s_i, Q(\mathcal{X}_i + \beta s_i)] \leq \mathcal{J}[\mathcal{X}_i, Q(\mathcal{X}_i)] + 10^{-4}\beta \mathcal{G}[\mathcal{X}_i, Q(\mathcal{X}_i)]^T s_i \tag{2.59}$$

$$|\mathcal{G}_i[\mathcal{X}_i + \beta s_i, Q(\mathcal{X}_i + \beta s_i)]| s_i^T \leq 0.9 |\mathcal{G}_i[\mathcal{X}_i, Q(\mathcal{X}_i)]^T s_i| \tag{2.60}$$

The line search is initialized with $\beta = 1$, and if this step size satisfies the strong Wolfe conditions, it is used. Otherwise, β is reduced according to an approximation of the one-dimensional function

$$\phi(\beta) = \mathcal{J}[\mathcal{X}_i + \beta s_i, Q(\mathcal{X}_i + \beta s_i)] \tag{2.61}$$

The step size is first reduced by taking the minimum of a quadratic approximation of ϕ. This minimum is

$$\frac{-\phi'(0)}{2[\phi(1)-\phi(0)-\phi'(0)]} \tag{2.62}$$

All subsequent iterations use the minimum of a cubic interpolation of ϕ. Using the most recent value of the step size, β_1, and second most recent value, β_2, we define

$$\begin{bmatrix} r_1 \\ r_2 \end{bmatrix} = \frac{1}{\beta_1-\beta_2} \begin{bmatrix} \frac{1}{\beta_1^2} & \frac{-1}{\beta_2^2} \\ \frac{-\beta_2}{\beta_1^2} & \frac{\beta_1}{\beta_2^2} \end{bmatrix} \begin{bmatrix} \phi(\beta_1)-\phi'(0)\beta_1-\phi(0) \\ \phi(\beta_2)-\phi'(0)\beta_2-\phi(0) \end{bmatrix} \tag{2.63}$$

and the step size is

$$\frac{-r_2+\sqrt{r_2^2-3r_1\phi'(0)}}{3r_1} \tag{2.64}$$

However, in the cubic approximation case, if the step size is greater than 0.5 or less than 0.1, it is set to that value respectively. This is necessary to reduce the step size at each iteration while bounding the error of the cubic approximation [27].

The Hessian is initialized to I. After the first step, before the update of the inverse Hessian, $\mathcal{H}$ is scaled by

$$\mathcal{H}_0 = \frac{\nu_i^T s_i}{\nu_i^T \nu_i} I \tag{2.65}$$

where

$$\nu_i = \mathcal{G}_{i+1} - \mathcal{G}_i \tag{2.66}$$

$$\tau_i = \mathcal{X}_{i+1} - \mathcal{X}_i \tag{2.67}$$

This is an approximation to one of the eigenvalues of the exact Hessian, and provides stability for the rest of the computation. The update at any step is then given by

$$\mathcal{H}_{i+1} = \mathcal{H}_i - \frac{\mathcal{H}_i\nu_i(\mathcal{H}_i\nu_i)^T}{\nu_i^T\mathcal{H}_i\nu_i} + \frac{\tau_i\tau_i^T}{\tau_i^T\nu_i} + \nu_i^T\mathcal{H}_i\nu_i(r_i r_i^T) \tag{2.68}$$

where

$$r_i = \frac{\tau_i}{\tau_i^T\nu_i} - \frac{\mathcal{H}_i\nu_i}{\nu_i^T\mathcal{H}_i\nu_i} \tag{2.69}$$

Chapter 3

Validation

The primal computational grid for all cases presented is a 3631 node triangular grid with 7052 cells around a NACA 0012 airfoil.

Although the number of parameters varies over the cases, the method used to parameterize the airfoil does not. After the airfoil is approximated by a 3^{rd} order spline, the parameters are chosen to be the y-components of the control points of all but the trailing-edge control point and the three leading-edge control points. This method fixes the chord length by fixing the leading- and trailing-edge points. The upper and lower surfaces have an equal number of control points, and the parameters are all initially scaled to 1.

In all cases, the finite-difference derivative and the adjoint derivative agreed to several decimal places.

3.1 Case 1

This case is designed to verify the optimizer using the objective function of Eq. 2.43. The airfoil is parameterized with 6 parameters, and the flow is at Mach 0.5 with an angle of attack of 5.0 degrees. The target pressure distribution is taken from the NACA0012 airfoil, and the shape is perturbed as demonstrated in Fig. 3.2. The optimizer is run on the perturbed shape.

The optimizer performs well, reducing the objective function 17 orders of magnitude and the derivative norm 9 orders in 200 iterations, as seen in Fig. 3.1. Fig. 3.2 shows that after a few iterations, the airfoil is almost visually identical to the target airfoil, and Fig. 3.3 shows that the same is true of the pressure distributions.

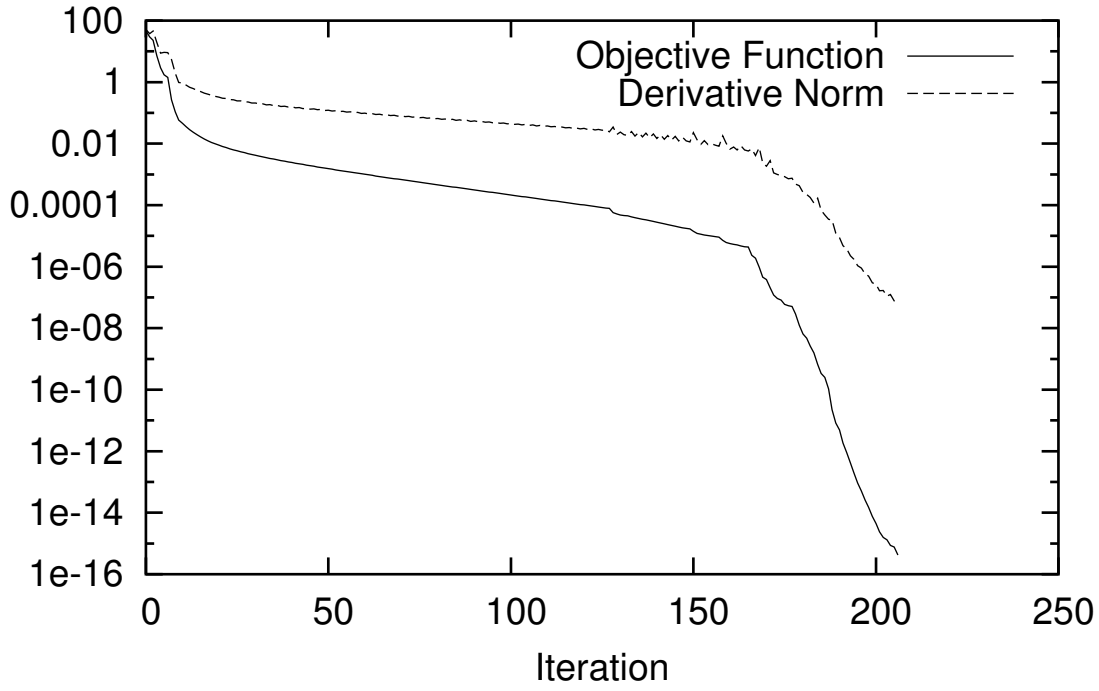

Figure 3.1: Objective Function and Derivative Norm Convergence for Case 1

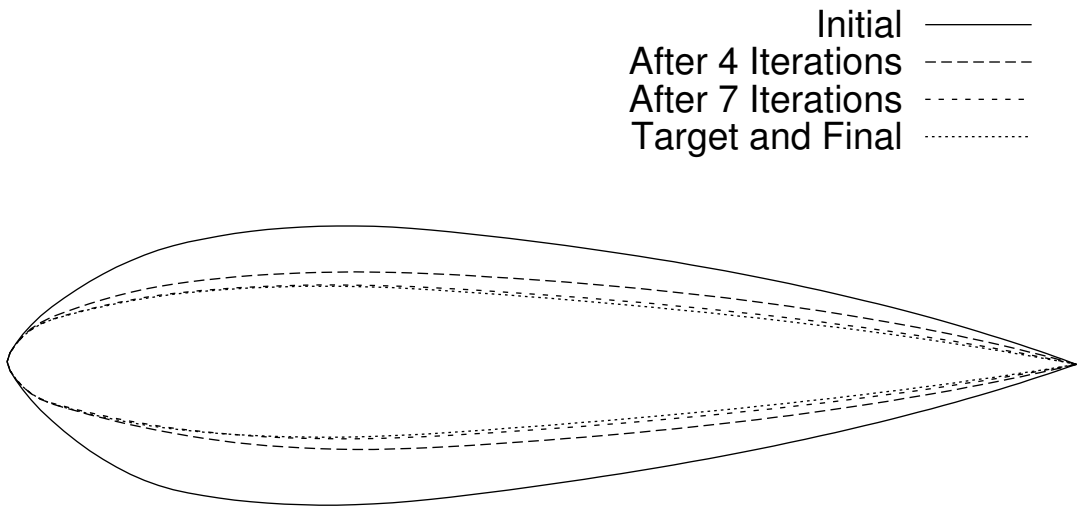

Figure 3.2: Various Airfoils for Case 1

3.2 Case 2

Using the same flow as in the case 1, but with 12 parameters, the minimizer is run with the objective function of Eq. 2.44. The drag and quarter-chord moment for the NACA 0012 are both quite small for this flow, and the target of the optimization is to increase the quarter-chord moment while keeping the drag coefficient low and the lift coefficient at its initial value.

The coefficients used, along with the optimal coefficients, are summarized in Table 3.1. The optimizer comes close to the target coefficients. Fig. 3.5 shows that the final airfoil is quite different from the initial and is overall quite an unusual shape. The case converges quickly, as Fig. 3.4 shows. Over 80 iterations, the derivative norm drops 4 orders of magnitude. The initial and final pressure distributions are given in Fig. 3.6.

	Lift	Drag	Moment
Weight	2.5	200	131
Target	3.010e-01	0.0	1.000e-01
Initial Value	3.010e-01	5.050e-03	7.673e-03
Optimal Value	3.069e-01	6.118e-03	9.950e-02

Table 3.1: Coefficients for Case 2

3.3 Case 3

When there is only one local optimum, as in the inverse case, the optimizer will find that. This case is designed to demonstrate the optimizer's ability to find this optimum from a number of starting positions when the optimum is at an unknown location. The objective function of Eq. 2.44 is used. C_M^* and C_L^* are respectively set to the C_M and C_L calculated from the flow over the unperturbed NACA 0012. C_D^* is set to 0.0, and the weights are set to put the most emphasis on drag minimization. The numerical values of the weights and coefficients for the function are given in Table 3.2. The NACA 0012 is parameterized using 4 parameters, and the flow is at Mach 0.8 with an angle of attack of 5.0 degrees. Under these conditions, a shock will be produced that will induce pressure drag. A successful optimization will thus be a reduction of the shock strength, or elimination of it entirely.

Three optimizations are performed. The first optimization is done on the unperturbed airfoil and is referred to as "Normal" in the figures. The second and third optimizations are done on airfoils which are respectively much thicker and thinner than the NACA 0012, and are referred to as "Thick" and "Thin" in the figures.

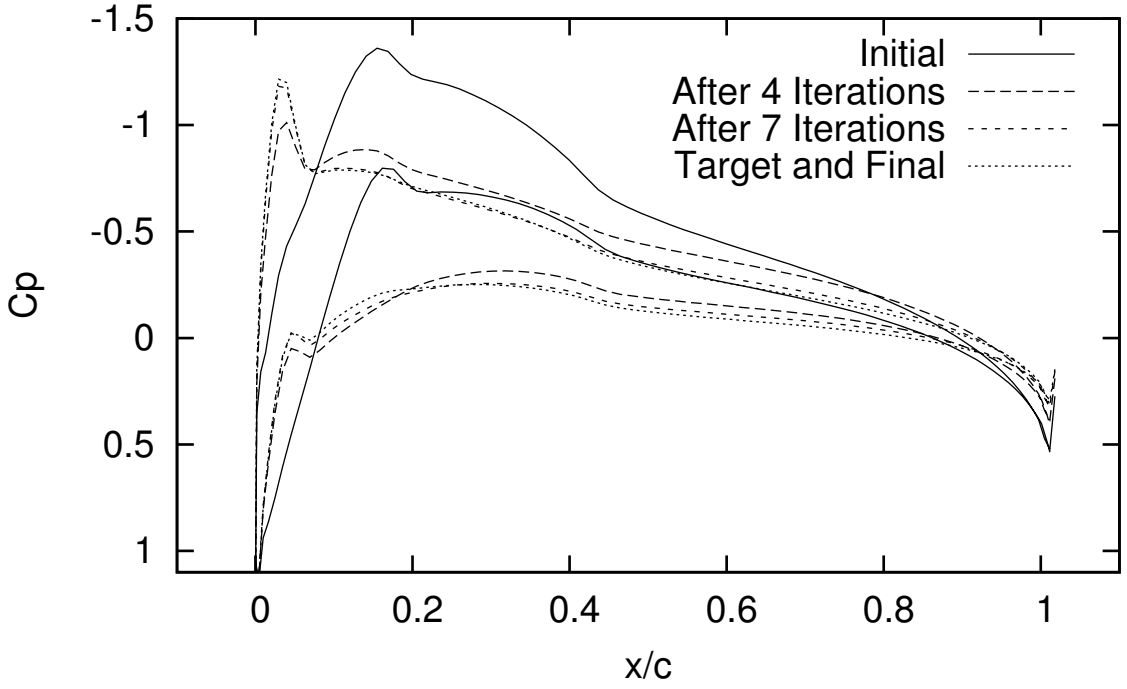

Figure 3.3: Various Pressure Distributions for Case 1

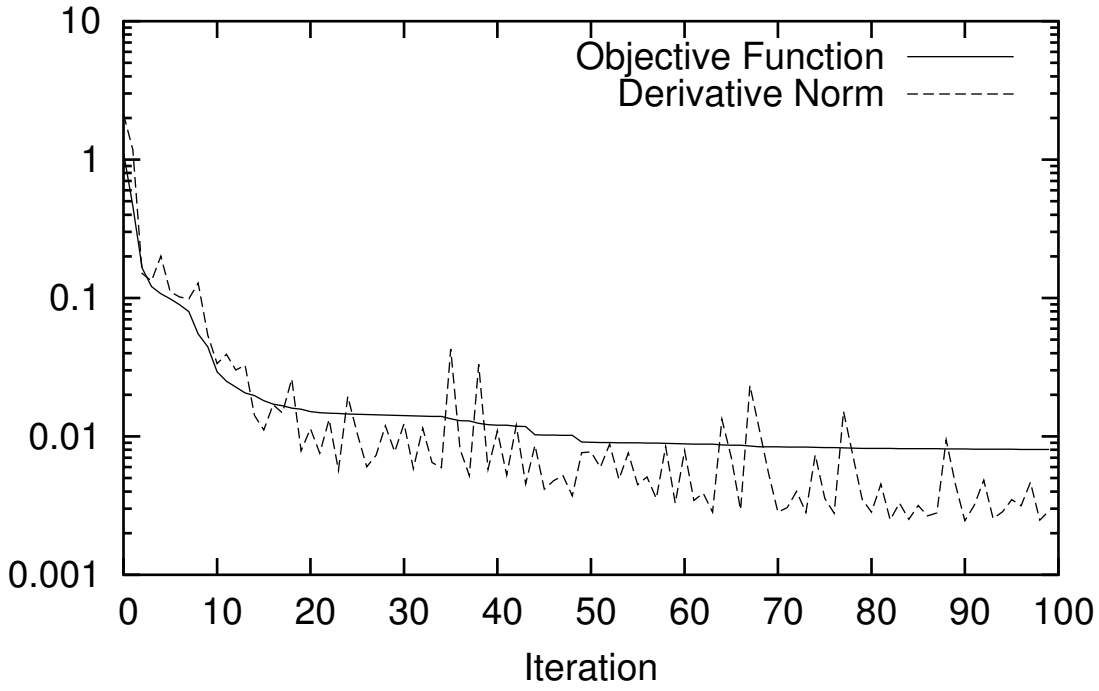

Figure 3.4: Objective Function and Derivative Norm Convergence for Case 2

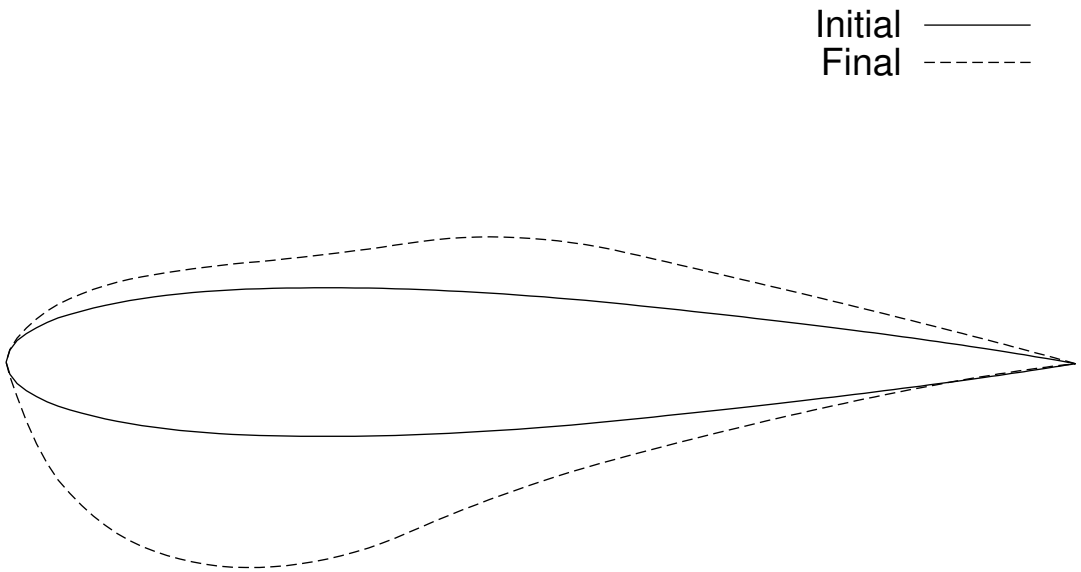

Figure 3.5: Initial and Final Airfoils for Case 2

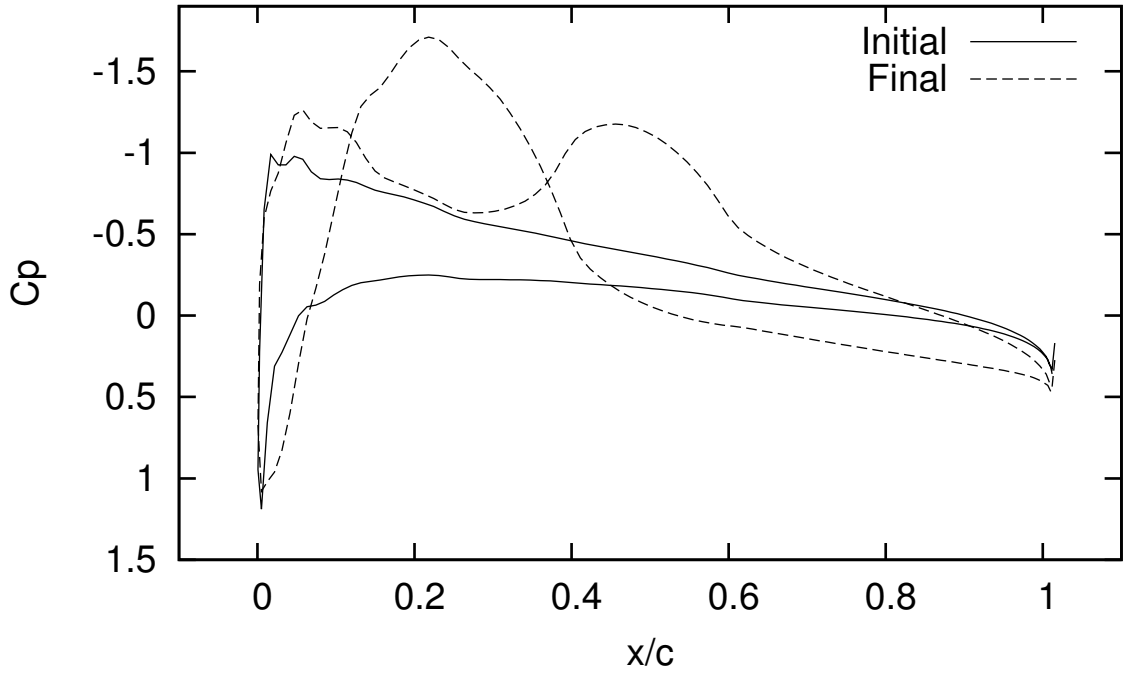

Figure 3.6: Initial and Final Pressure Distributions for Case 2

All cases converge to the same optimum. The shape of the airfoils and the optimal airfoil is shown in Fig. 3.7. The optimal airfoil is a very strange one, and presumably so because only the Euler equations are solved with no structural constraints enabled. Figs. 3.8 and 3.9 shows that within 30 iterations, all cases essentially converge to the same objective function value and are approaching a local optimum, although the thicker airfoil case takes longer to converge. Presumably, this is because it starts with a much stronger shock, as can be seen in Fig. 3.10, and is thus further from the optimum. Also, the convergence of the thicker airfoil case stalls during iterations 10 to 15. Fig. 3.11 shows that after it recovers from that, similar numbers of flow solves and derivative calculations are required for all cases. In general, 4 flow solves and 2 derivative calculations are required per minimizer iteration after startup.

	Lift	Drag	Moment
Weight	100	2500	10
Target	8.803e-01	0.0	1.301e-01
Initial Value Normal	8.803e-01	1.303e-01	1.301e-01
Initial Value Thin	7.921e-01	1.141e-01	5.506e-02
Initial Value Thick	5.172e-01	1.838e-01	9.242e-02
Optimal Value	7.329e-01	9.371e-02	3.078e-02

Table 3.2: Coefficients for Case 3

3.4 Case 4

This case is almost identical to the previous one. This time, however, the angle of attack is a allowed to vary, and constraints are added to prevent the airfoil from becoming too thin. Four constraints in the form of Eq. 2.45 are applied to the grid points located on the upper and lower surface at approximately 10, 25, 50 and 90 percent chord. A summary of the values associated with these constraints is given in Table 3.3. Figs. 3.12 and 3.13 give the convergence for all the airfoils. In general, convergence took longer than case 3. This is probably due to the additional parameter, the angle of attack, as it must vary greatly from its initial value of 5 degrees to its final value of 0.3 degrees. The optimal objective function value is lower than that of the unconstrained case, and this is solely due to the angle of attack varying. The optimal airfoil activates three of the penalty constraints. The other constraint is only active during the optimization. The aerodynamic coefficients for this case are summed up in Table 3.4. The differences in pressure distribution and airfoil shape for the two cases are given in Fig. 3.15 and Fig. 3.14 respectively. Since the angle of attack is free, the optimizer is able to reduce the

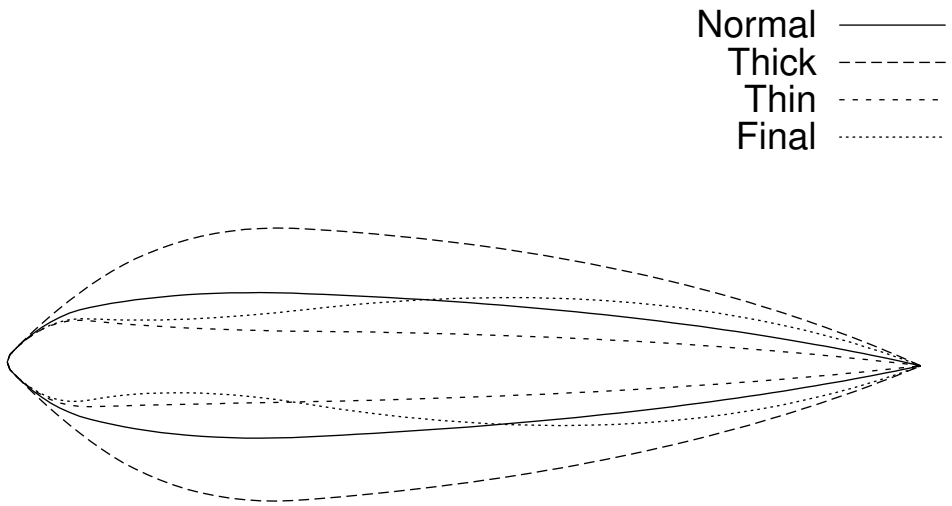

Figure 3.7: Initial and Final Airfoils for Case 3

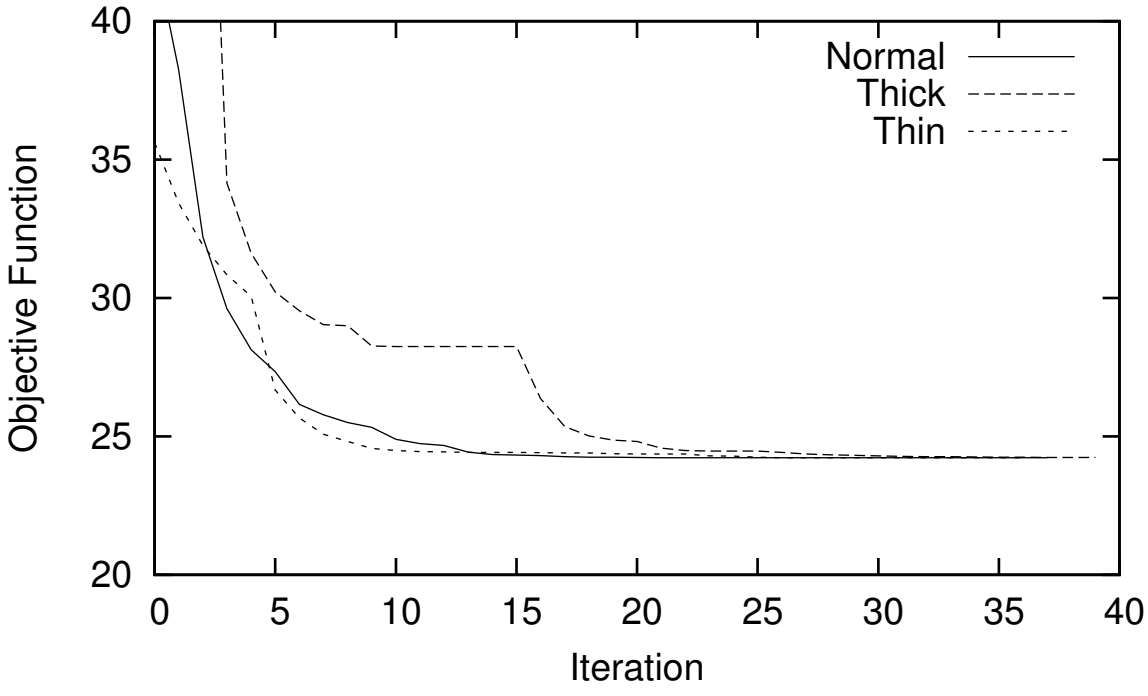

Figure 3.8: Objective Function Convergence for Case 3

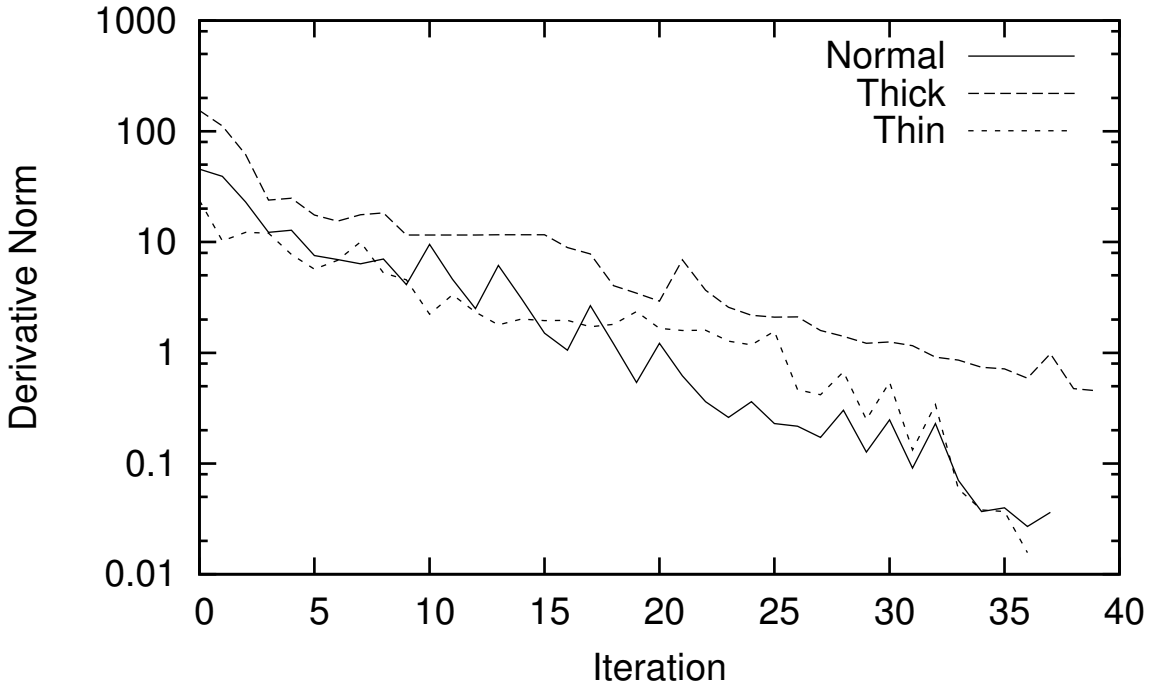

Figure 3.9: Derivative Norm Convergence for Case 3

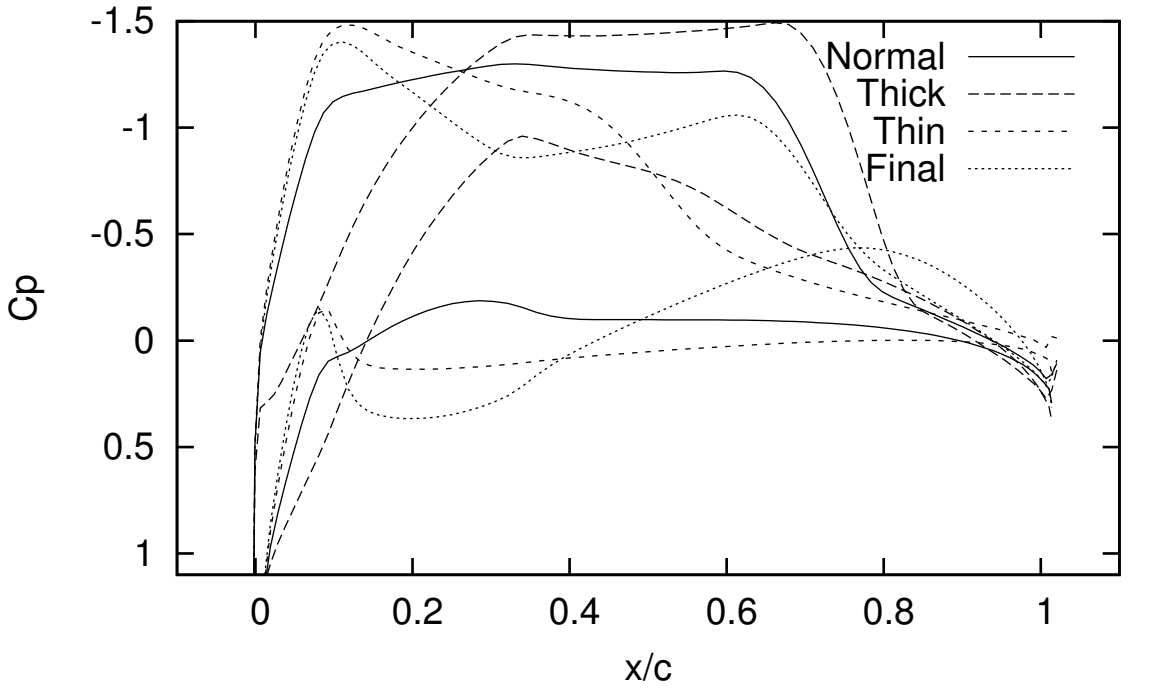

Figure 3.10: Initial and Final Pressure Distributions for Case 3

angle of attack and add camber, producing a more optimal airfoil than in case 3. The airfoil produced by this case is a much more reasonable airfoil.

Percent Chord	10	25	50	90
Scale	1.0e-4	1.0e-3	1.0e-3	1.0e-4
Minimum Distance	8.2e-2	9.0e-2	3.0e-2	1.0e-2
Starting Distance	8.696e-2	1.208e-1	1.037e-1	3.316e-2
Optimal Distance	7.791e-2	8.278e-2	5.131e-2	8.369e-3
Optimal Value	1.666e-1	5.212e-2	0.0	2.659e-2

Table 3.3: Constraint Summary for Case 4

	Lift	Drag	Moment
Weight	100	2500	10
Target	8.803e-01	0.0	1.301e-01
Unconstrained Optimal Value	7.329e-01	9.371e-02	3.078e-02
Constrained Optimal Value	7.742e-01	5.194e-02	2.452e-01

Table 3.4: Coefficients for Case 4

3.5 Case 5

This case explores the effect of the number of parameters on the convergence of the optimizer and the optimum value found. The flow and objective function are identical to case 3. Optimizations are performed with 4, 6, 8, and 10 parameters, and all start with the NACA 0012 shape.

Fig. 3.16 shows the final airfoils for all cases. Again, without structual contraints or more accurate fluid modelling, these airfoils are bound to be physically unfeasible. With greater numbers of parameters, the derivative norm converges more slowly. However, within 15 iterations, as seen in Figs. 3.17 and 3.18, all cases converge their objective functions and greater numbers of parameters enable a substantially lower objective function value. All the coefficients for this case are summarized in Table 3.5. Most of this reduction in the objective function came from the reduction in drag produced by the reduction of the shock. as seen in Fig. 3.19.

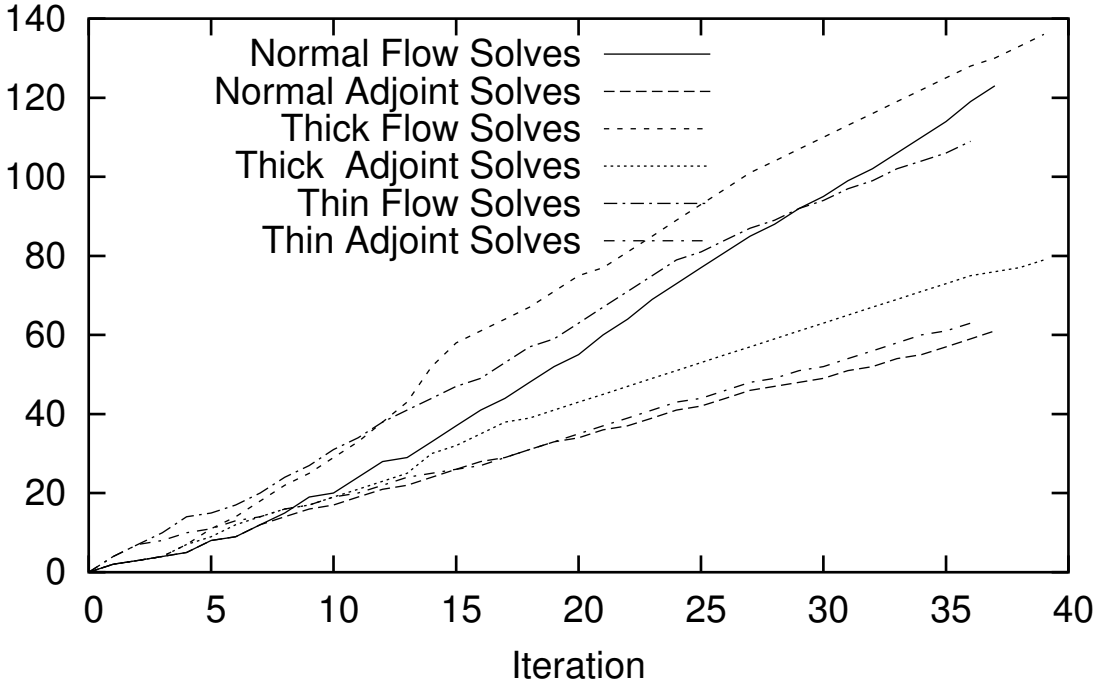

Figure 3.11: Numbers of Flow Solves and Derivative Solves for Case 3

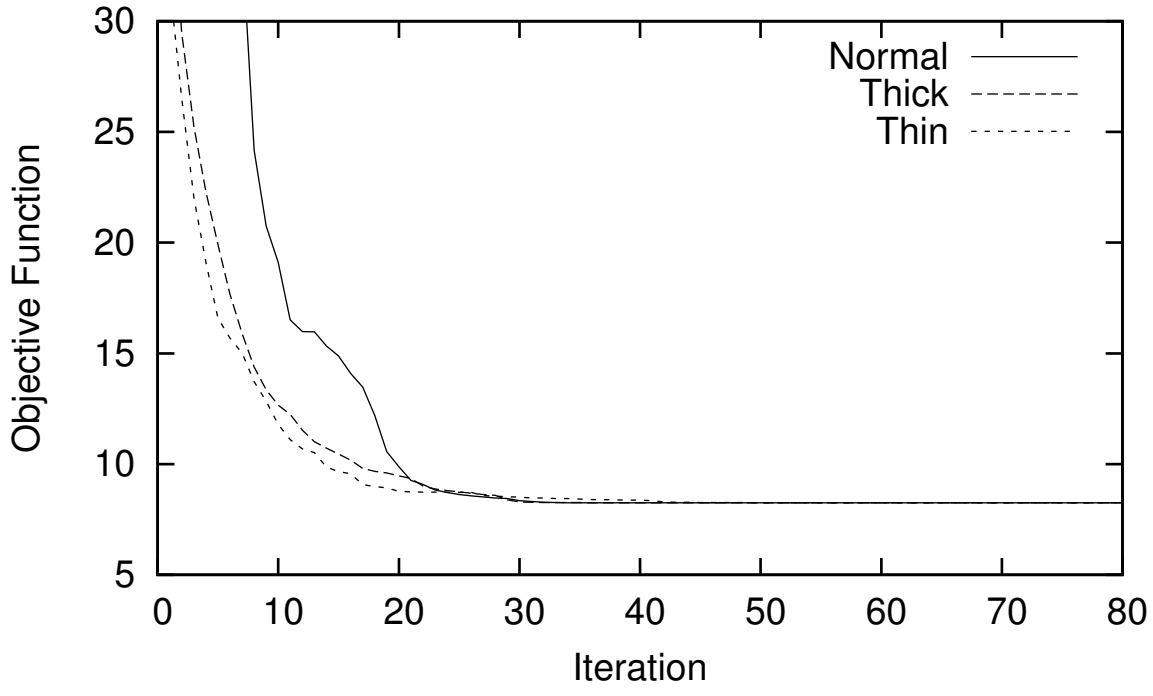

Figure 3.12: Objective Function Convergence for Case 4

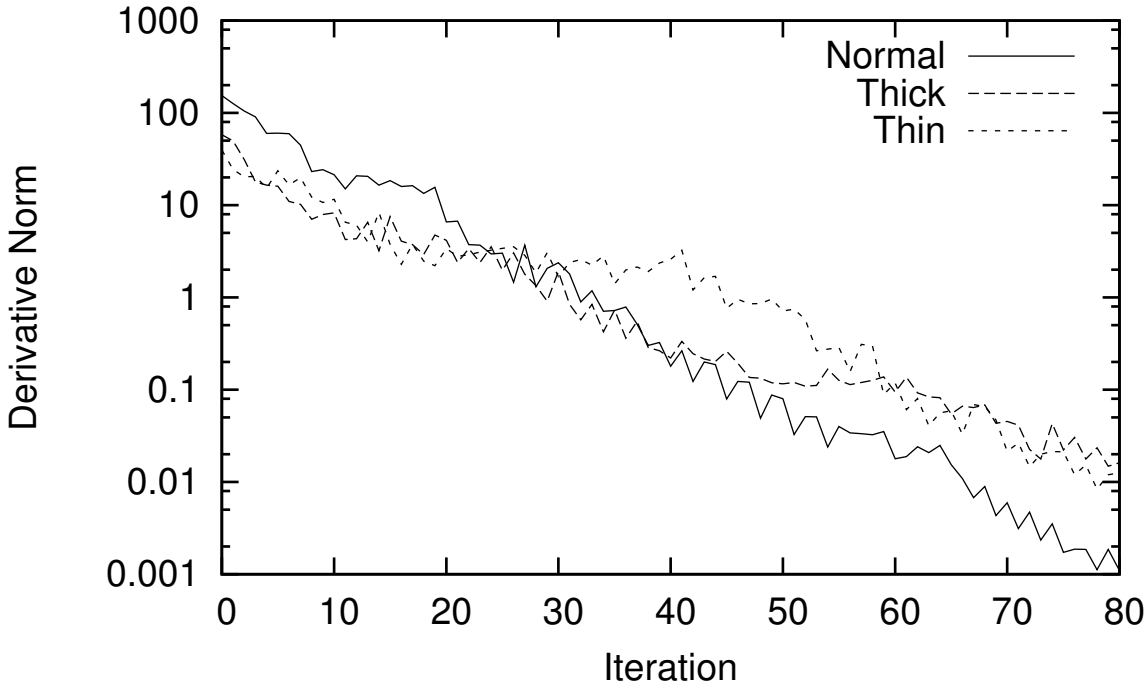

Figure 3.13: Derivative Norm Convergence for Case 4

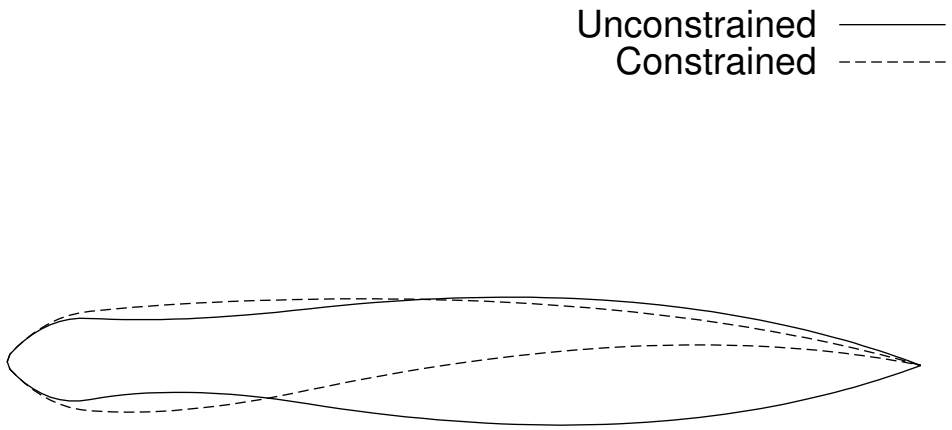

Figure 3.14: Optimal Airfoils for Cases 3 and 4

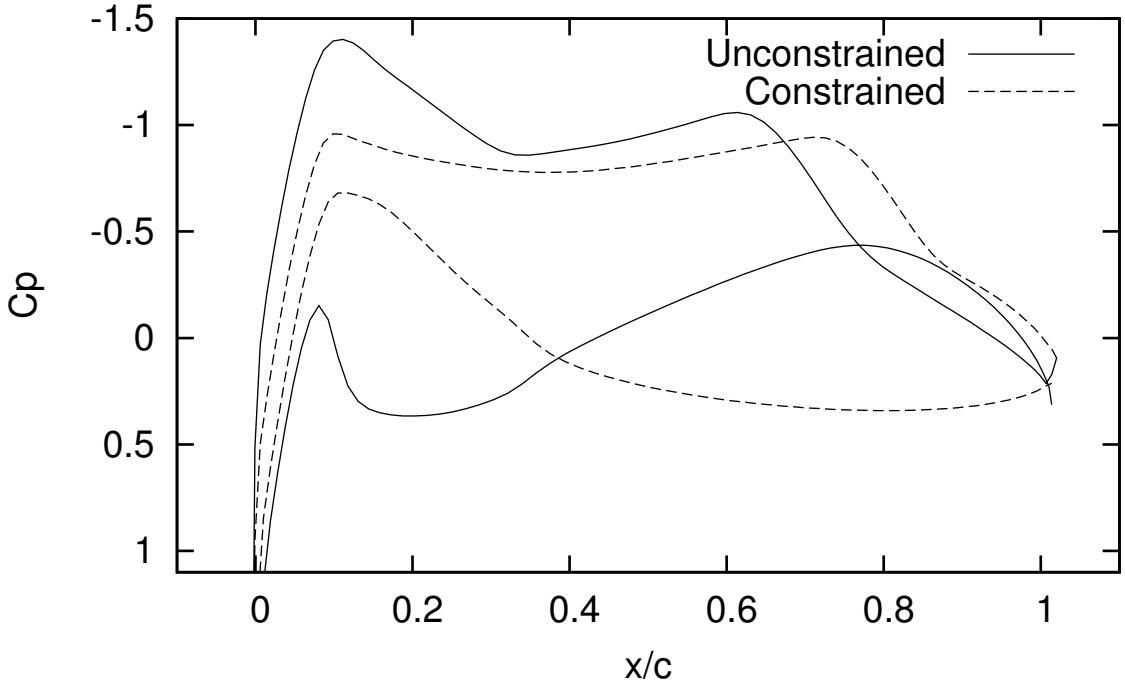

Figure 3.15: Optimal Pressure Distributions for Cases 3 and 4

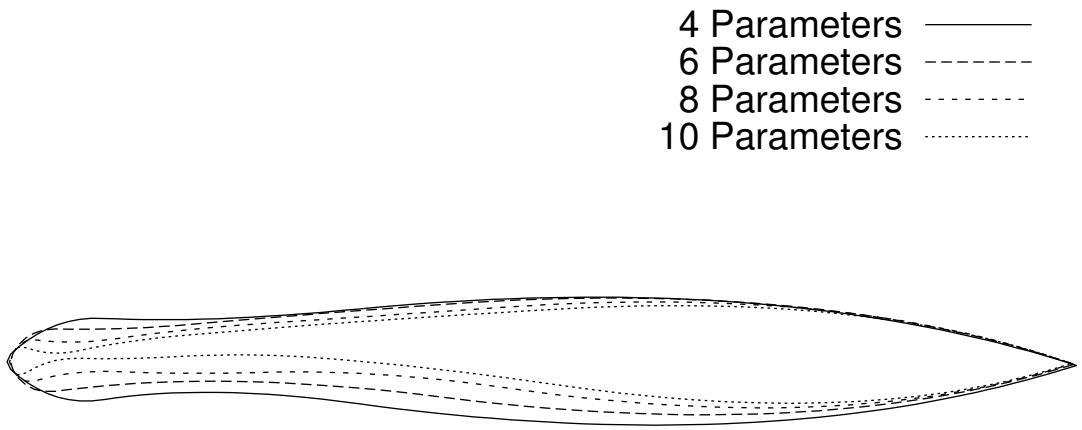

Figure 3.16: Final Airfoils for Case 5

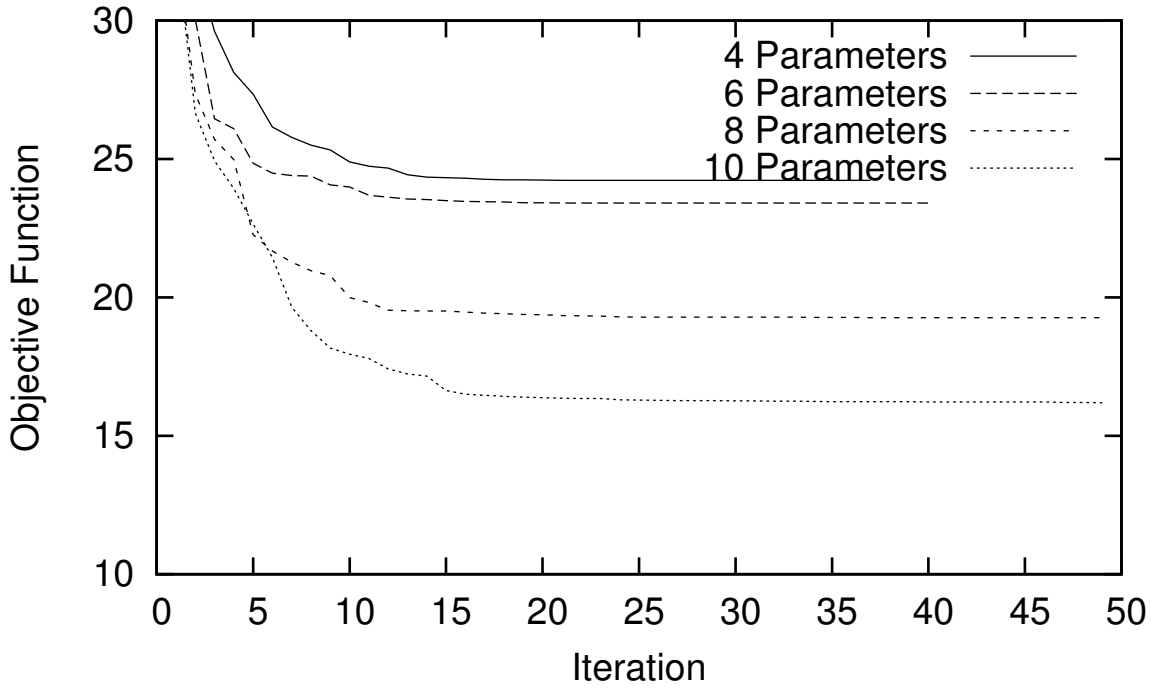

Figure 3.17: Objective Function Convergence for Case 5

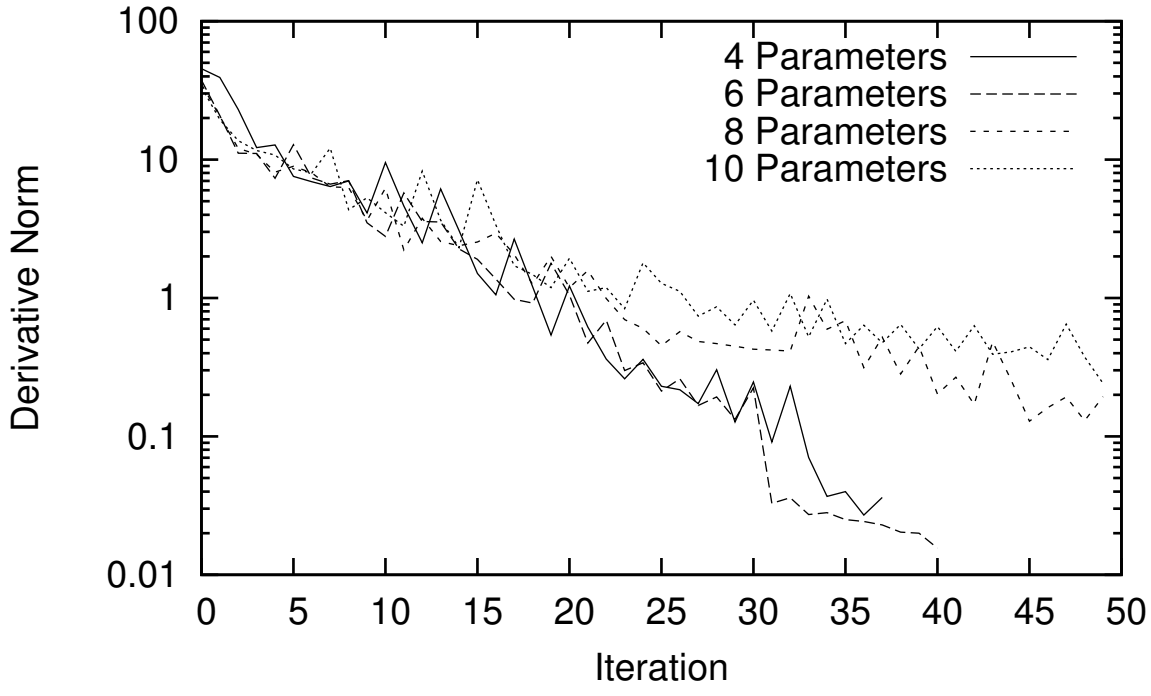

Figure 3.18: Derivative Norm Convergence for Case 5

	Lift	Drag	Moment
Weight	100	2500	10
Target	8.803e-01	0.0	1.301e-01
4 Parameter Optimal Value	7.329e-01	9.371e-02	3.078e-02
6 Parameter Optimal Value	7.370e-01	9.221e-02	3.190e-02
8 Parameter Optimal Value	7.624e-01	8.431e-02	3.121e-02
10 Parameter Optimal Value	7.836e-01	7.786e-02	2.812e-02

Table 3.5: Coefficients for Case 5

3.6 Case 6

This case is identical to case 5, except that the angle of attack is allowed to vary, and penalty constraints, the same as in case 4, are added to keep the airfoil reasonable. Convergence of the runs is given in Figs. 3.20 and 3.21. The optimal objective function for all runs is lower than the corresponding objective function for the unconstrained cases. The optimal airfoils are all just inside the penalty region, as seen in Table 3.6. Airfoil shape and pressure distribution data are given in Figs. 3.22 and 3.23 respectively, and the aerodynamic coefficients and optimal angle of attack for each run are summarized in Tables 3.7 and 3.8. Generally, a greater number of parameters improves the values of all coefficients and the objective function, although the seven parameter case seems not to do as well as the five parameter case. Because the parameterizations are not subsets of one another, the seven parameter case cannot describe the same shapes as the five parameter case, and for this objective function, the five parameter case is more optimal. In all cases, the angle of attack was lowered with more parameters.

Percent Chord	10	25	50	90
5 Parameter Optimal Value	1.666e-1	5.212e-2	0.0	2.659e-2
7 Parameter Optimal Value	3.627e-1	2.831e-3	0.0	6.374e-2
9 Parameter Optimal Value	5.089e-1	6.873e-4	0.0	6.000e-2
11 Parameter Optimal Value	6.531e-1	2.112e-4	0.0	5.287e-2

Table 3.6: Constraint Summary for Case 6

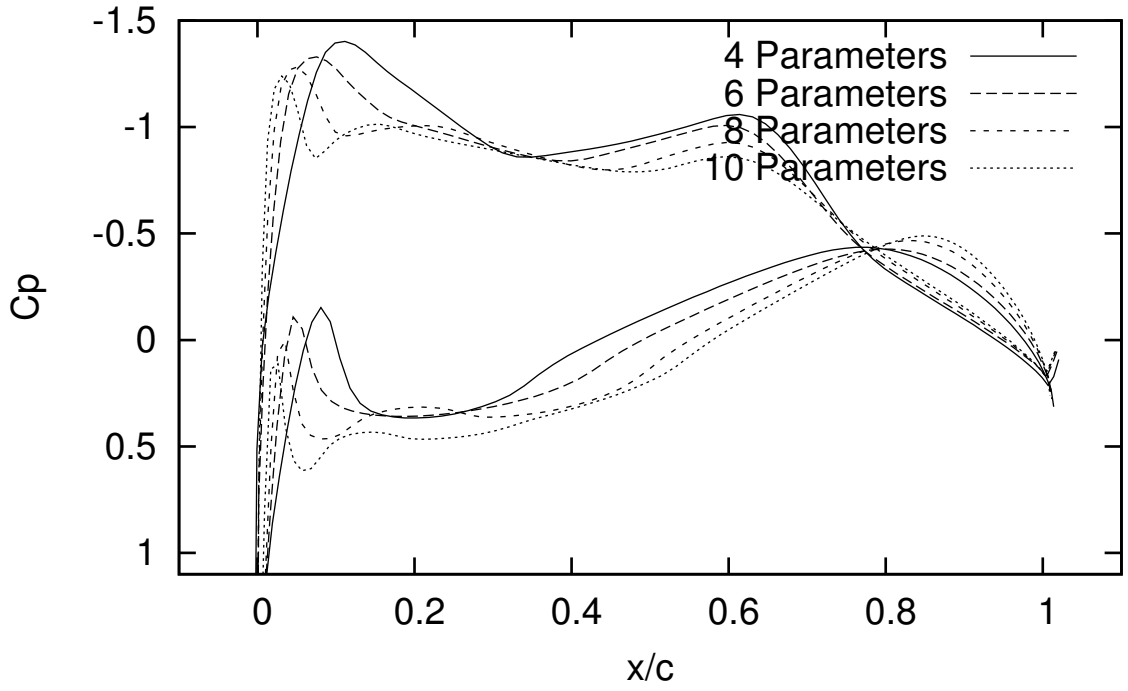

Figure 3.19: Final Pressure Distributions for Case 5

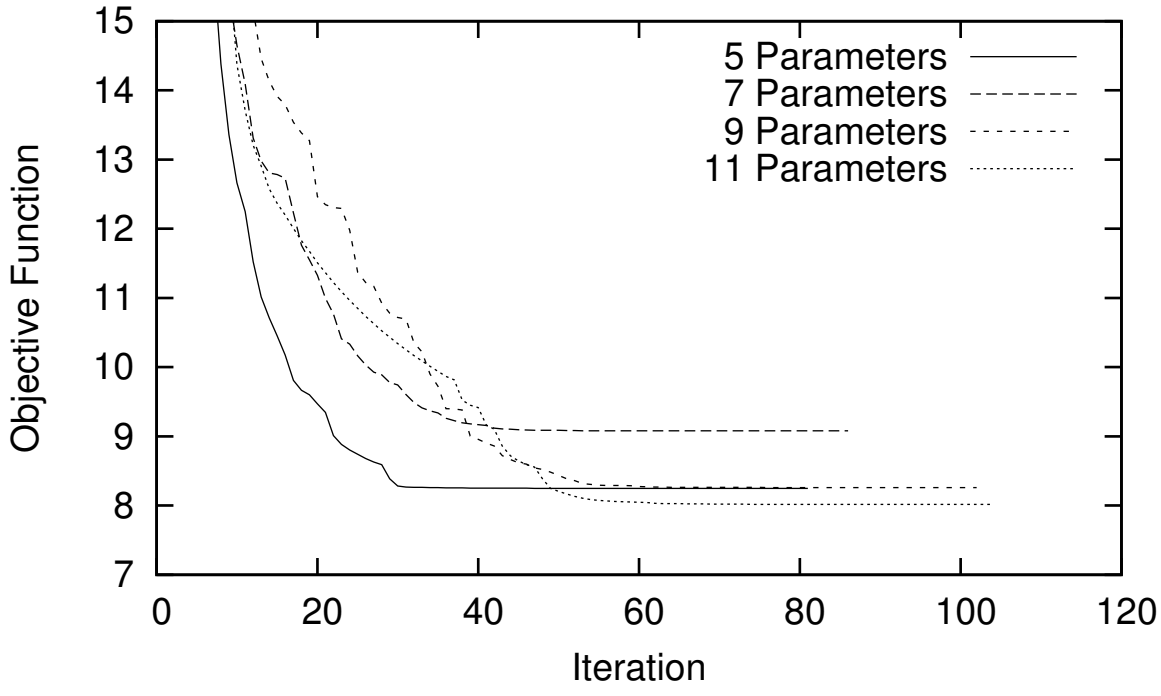

Figure 3.20: Objective Function Convergence for Case 6

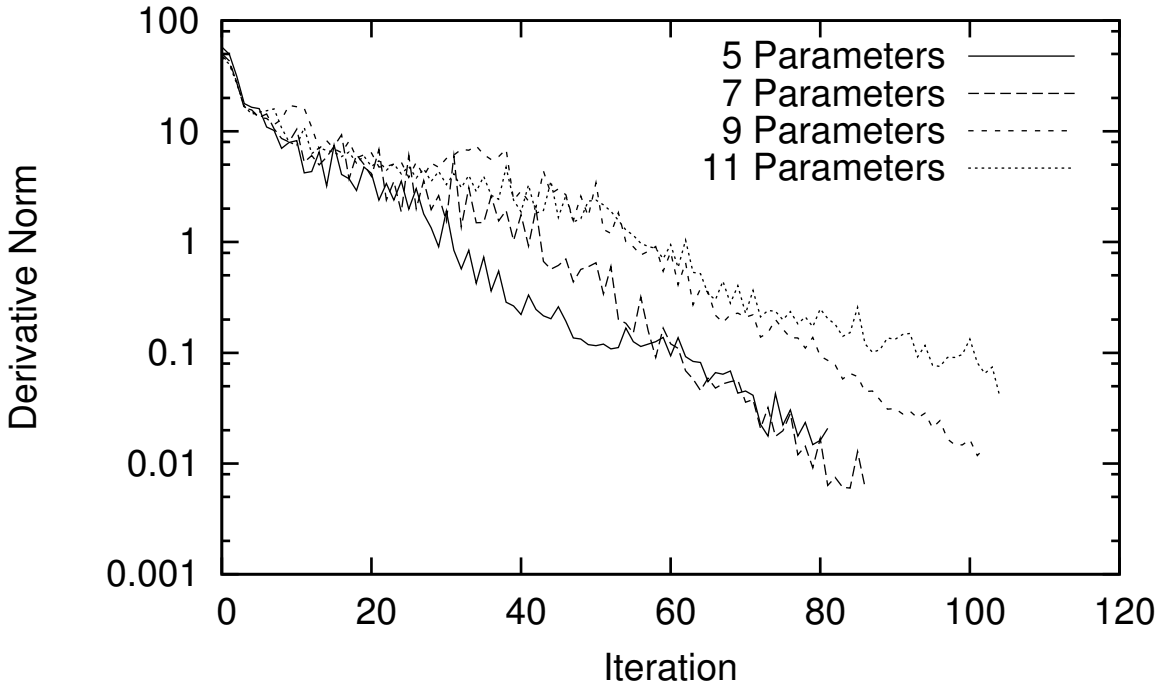

Figure 3.21: Derivative Norm Convergence for Case 6

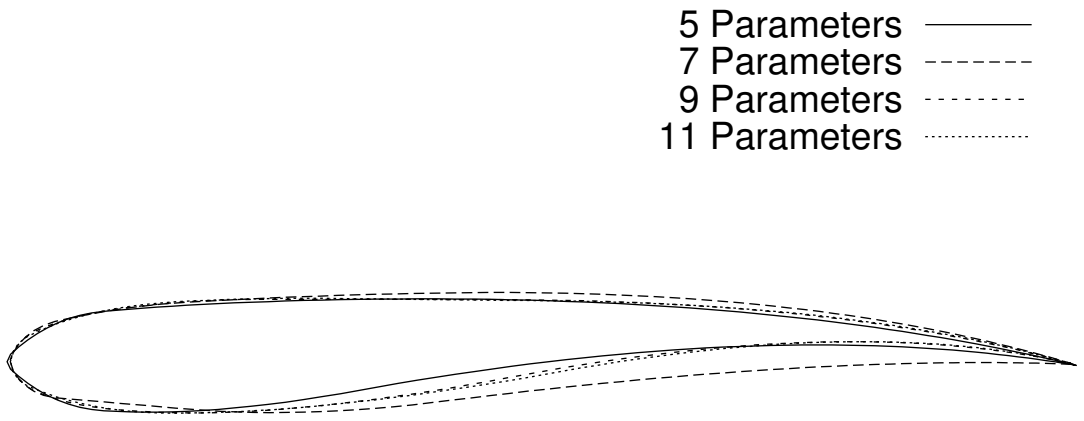

Figure 3.22: Final Airfoils for Case 6

Parameters	Angle of Attack
5	0.3029
6	0.1500
9	0.1177
11	0.0679

Table 3.7: Optimal Angle of Attack for Case 6

	Lift	Drag	Moment
Weight	100	2500	10
Target	8.803e-01	0.0	1.301e-01
5 Parameter Optimal Value	7.742e-01	5.194e-02	2.452e-01
7 Parameter Optimal Value	7.676e-01	5.376e-02	2.537e-01
8 Parameter Optimal Value	7.739e-01	5.057e-02	2.585e-01
11 Parameter Optimal Value	7.770e-01	4.927e-02	2.615e-01

Table 3.8: Coefficients for Case 6

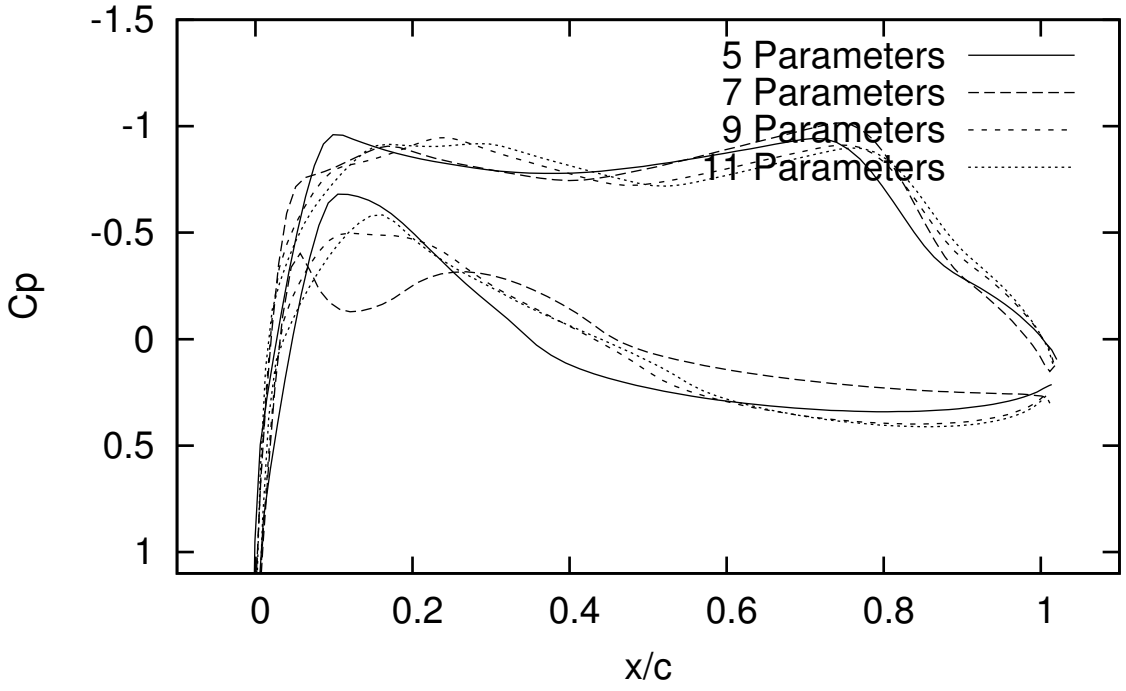

Figure 3.23: Final Pressure Distributions for Case 6

Chapter 4

Conclusions and Directions for Future Work

An efficient optimization method has been presented for the Euler equations on unstructured grids. It has been verified with a number of objective functions over a number of flow regimes. In translating the method from structured to unstructured grids, a fast grid deformation method has been developed that creates a differentiable grid, which is a requirement of this particular optimization method. Finally, the goal of the paper is achieved; the present unstructured version of the optimization method has similar costs to the structured version [22, 21].

There are several areas that could be the target of future work. The flow solver on top of which the optimizer is built already has the ability to solve laminar and turbulent cases in 2D and laminar cases in 3D [14]. Extending the adjoint solver to any of these regimes is not difficult.

The segment-spring analogy has been used to deform viscous grids, but the modifications to it that ensure grid differentiability have not been tested with viscous grids. If the new method fails, another differentiable method will have to be found. The grid-deformation method is completely independent of every other major part of the optimizer, and this allows it to be interchanged with a better one should one arise. For instance, the torsion spring analogy, which couples the spatial dimensions of the grid points, can be made differentiable, and this may be necessary for viscous and 3D grids.

The parameterization method in 3D can be a difficult problem. For simple geometries, the present method should work. However, geometries that come directly from a CAD environment, which provides the most flexible parameterizations, generally require at least boundary-layer regridding when parameters vary. This eliminates differentiability, and thus reduces the accuracy of the derivatives being used by the BFGS algorithm. Perhaps a more fault tolerant algorithm would have to replace the BFGS algorithm for these parameterizations.

No attempt was made to parallelize any of the algorithms because the flow solver is not parallelized and has no grid distribution capabilities. Effective preconditioning techniques for the parallel Newton-Krylow algorithm, the adjoint, and the grid deformation algorithm are needed. However, since basically the same method is effective in solving linear systems throughout the present work, one such technique would likely fulfill all needs.

Bibliography

[1] Batina, J. T. Unsteady Euler airfoil solutions using unstructured dynamic meshes. *AIAA Journal 28*, 8 (1990), 1381–1388.

[2] Blom, F. J. Considerations on the spring analogy. *International Journal for Numerical Methods in Fluids 32* (2000), 647–668.

[3] Coello, C. A. C. *An Empirical Study of Evolutionary Techniques for Multiobjective Optimization in Engineering Design*. PhD thesis, Tulane University, New Orleans, April 1996.

[4] Cuthill, E., and McKee, J. Reducing the bandwidth of sparse symmetric matrices. In *24th National Conference of the Association for Computing Machinery* (New York, 1969), no. ACM P-69, Brandon Press, pp. 157–172.

[5] de Boor, C. *A Practical Guide to Splines*. Springer–Verlag, New York, 1978.

[6] Elliot, J., and Peraire, J. Aerodynamic design using unstructured meshes. AIAA Paper 96–1941, June 1996.

[7] Elliott, J., and Peraire, J. Aerodynamic optimization on unstructured meshes with viscous effects. AIAA Paper 97–1849, June 1997.

[8] Gatsis, J., and Zingg, D. W. A fully-coupled algorithm for aerodynamic design optimization. In *48th Annual CASI Conference* (Toronto, 2001), pp. 227–236.

[9] Giles, M. B., and Pierce, N. A. An introduction to the adjoint approach to design. *Flow, Turbulence and Combustion 65*, 3/4 (2000), 393–415.

[10] Goldberg, D. E. *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison–Wesley, Reading, 1989.

[11] Gunzburger, M. D. Introduction into mathematical aspects of flow control and optimization. In *Inverse Design and Optimization*, R. V. den Braembussche and M. Manna, Eds., von Karman Institute for Fluid Dynamics Lecture Series. Belgium, 1997.

[12] JAMESON, A., SCHMIDT, W., AND TURKEL, E. Numerical solutions of the Euler equations by finite volume methods using Runge-Kutta time stepping. AIAA Paper 81–1259, June 1981.

[13] LI, D., AND FUKUSHIMA, M. A modified BFGS method and its global convergence in nonconvex minimization. Tech. rep., Hunan University, January 1998.

[14] MANZANO, L. M., LASSALINE, J. V., AND ZINGG, D. W. A Newton–Krylov algorithm for the Euler equations using unstructured grids. 41st Aerospace Sciences Meeting, AIAA 2003-0274, AIAA.

[15] MARTINS, J. R. R. A., KROO, I. M., AND ALONSO, J. J. An automated method for sensitivity analysis using complex variables. AIAA Paper 2000-0689, January 2000.

[16] NADARAJAH, S. K., AND JAMESON, A. A comparison of the continuous and discrete adjoint approach to automatic aerodynamic optimization. AIAA 38th Aerospace Sciences Meeting and Exhibit, AIAA-2000-0667.

[17] NADARAJAH, S. K., AND JAMESON, A. Studies of the continuous and discrete adjoint approach to viscous automatic aerodynamic optimization. AIAA 15th Computational Fluid Dynamics Conference, AIAA-2001-2530.

[18] NEMEC, M. *Optimal Shape Design Of Aerodynamic Configurations: A Newton–Krylov Approach*. PhD thesis, University of Toronto Institute for Aerospace Studies, 2003.

[19] NEMEC, M., AND ZINGG, D. W. Aerodynamic shape optimization using the discrete adjoint method. In *48th Annual CASI Conference* (Toronto, 2001), pp. 203–213.

[20] NEMEC, M., AND ZINGG, D. W. Multi-point and multi-objective aerodynamic shape optimization. *AIAA Journal Submission* (November 2002).

[21] NEMEC, M., AND ZINGG, D. W. Multi-point and multi-objective aerodynamic shape optimization. 9th AIAA/ISSMO Symposium, AIAA 2002-5548.

[22] NEMEC, M., AND ZINGG, D. W. A Newton–Krylov algorithm for aerodynamic design using the Navier–Stokes equations. *AIAA Journal 40*, 6 (June 2002), 1146–1154.

[23] NEMEC, M., ZINGG, D. W., AND PULLIAM, T. H. Multi-point and multi-objective aerodynamic shape optimization. AIAA Paper 2002–5548, Atlanta, September 2002.

[24] NIELSEN, E. J., AND ANDERSON, W. K. Aerodynamic design optimization on unstructured meshes using the Navier Stokes equations. *AIAA Journal 37*, 11 (1999), 1411–1419.

[25] NIELSON, E. J., AND ANDERSON, W. K. Recent improvements in aerodynamic design optimization on unstructured meshes. *AIAA Journal 40*, 6 (2002).

[26] NOCEDAL, J., AND WRIGHT, S. J. *Numerical Optimization*. Springer–Verlag, New York, 1999.

[27] PRESS, W. H., TEUKOLSKY, S. A., VETTERLING, W. T., AND FLANNERY, B. P. *Numerical Recipes in C*. Cambridge University Press, New York, 1992.

[28] SAAD, Y., AND SCHULTZ, M. H. GMRES: A generalized minimal residual algorithm for solving nonsymmetric linear systems. *SIAM Journal on Scientific and Statistical Computing 7*, 3 (1986), 856–869.

[29] SAMAREH, J. A. Survey of shape parametrization techniques for high-fidelity multidisciplinary shape optimization. *AIAA Journal 39*, 5 (2001), 877–883.

[30] SHENOY, A., HEINKENSCHLOSS, M., AND CLIFF, E. M. Airfoil design by an all-at-once method. Rice University Center for Parallel Computation CRPC–TR97703–S, November 1997.

[31] WEHNER, E. A Newton–Krylov solver for the Euler equations on unstructured grids. Master's thesis, University of Toronto Institute for Aerospace Studies, 2001.

[32] WRENN, G. A. An indirect method for numerical optimization using the Kreisselmeier-Steinhauser function. NASA CR–4220, March 1989.

[33] ZINGG, D. W., NEMEC, M., AND CHISHOLM, T. A Newton–Krylov algorithm for aerodynamic analysis and design. Keynote paper, International Conference on Computational Fluid Dynamics (ICCFD) 2, Sydney, July 2002.