Variable Complexity Optimization

by

Praveen Thokala

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

Abstract

Variable Complexity Optimization

Praveen Thokala

Master of Applied Science

Graduate Department of Aerospace Science and Engineering

University of Toronto

2005

This thesis deals with variable-complexity optimization, which consists of either using high and low fidelity models of the analysis or using variable parameterization of the optimization problem to reduce the computational cost of the optimization process. In this thesis, we present a variable-fidelity framework based on the approach proposed by Alexandrov that is mathematically robust. We then present the results on analytical test cases for the framework and a variable parameterization method, which involves using different design variables during the course of the optimization process. We then present our results for a 2-D airfoil optimization problem.

Acknowledgements

I would like to thank Prof. Joaquim Martins for his excellent teaching, guidance and supervision. Without his readiness to help me out with any problems that I had and his passion for research, I would not have done this. I would also like to thank Prof. David Zingg for letting me use OPTIMA-2D for my research.

I would like to thank my lab mates Alan and Nicholas, for helping me whenever I needed and for being so much fun. I learnt a lot from them. I would also like to thank Scott, Nathan and Laurent for a professional, yet enjoyable atmosphere in the lab.

Finally, I would like to thank Robert, Ravi, Fred, Jon, Azhar, Udit, Ming, Manish, Jai, Francisco, CFD group, Space Flight group, Combustion group and all the people of UTIAS for a great time at the Institute.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Motivation

The enormous computational cost of optimization using high-fidelity solvers can make the process prohibitively expensive. The optimization process usually requires many function evaluations, especially when gradients are needed and calculated by finite differencing. Therefore, for practical design, savings in computational time can be made by running the optimization on a low-fidelity solver. The optimum design can be validated by a high fidelity model and corrections are often applied in an iterative process. This approach of the combined usage of both high-fidelity and low-fidelity models is known as variable fidelity optimization.

A similar approach, which is based on changing the design variables during the course of optimization problem, can be used to reduce the computational time. This method has not been explored to its fullest, as far as we know, so we intend to explore this method and from now onwards we will call it the variable parameterization method.

1.2 Variable Complexity Methods

Variable complexity optimization strategies can be divided into methods that use variable fidelity of the analysis and methods that use variable parameterization of the problem. The difference is briefly explained in this section.

1.2.1 Variable Fidelity Method

Variable fidelity method can be described as an optimization process that is performed using a low-fidelity solver and is validated by a high fidelity model. Corrections are often applied in an iterative process. This approach of the combined usage of both high-fidelity and low-fidelity models is known as variable fidelity optimization. The different types of low fidelity models are:

- Simplified physics model

  –Different physical models can be used to describe a problem. For example, in aerodynamics, Navier–Stokes equations for nonlinear viscous flow can be simplified with linear potential models that describe inviscid, irrotational and incompressible flow.

- Variable-resolution model

  –A single physical model can be computed on meshes of varying degree of refinement. For example, a CFD solution can be evaluated using a coarser grid instead of a fine grid.

- Variable accuracy models

  –Single numerical model converging to a varying degree of accuracy (converge analysis to user specified tolerance). For example, all aerodynamic solver codes are usually equipped with a user–specific convergence criterion which can be decreased to speed up the process.

- Data fitting models

  –Low fidelity model can be based on fitting surfaces to samples of high–fidelity responses. For example, classical response surface models, splines or kriging models can be used.

1.2.2 Variable Parameterization

When using variable parameterization, the design variables are changed during the course of the optimization process. We start of with fewer design variables and optimize the discretized problem and then perform an accurate optimization with more design variables. This method is analogous to the multigrid methods in CFD. This method is more relevant for the problems which have a smooth variation of design variables, i.e., when the objective function is smooth with respect to the design variables. This method works well with problems like shape opti-

mization where the optimum is similar for different sets of design variables i.e. if they have the same physical meaning. For example, in case of an airfoil optimization (a problem where the design variables control the shape) the optimum (airfoil shape which satisfies the target criteria) would be similar for different sets of design variables, say for 10 and 15 design variables which control the shape and which are equally distributed over the airfoil. The design variables can then be changed during the course of the design process. For example, in case of aerodynamic optimization problems like airfoil optimization, rotor-craft optimization, we can start off with a few design variables, get an idea of the problem and use engineering intuition to solve the optimization problem in an efficient manner with more design variables.

1.3 Review of Variable Fidelity Optimization Algorithms

Variable fidelity optimization schemes have been devised by many researchers in various forms. A survey conducted a decade ago on structural optimization by Barthelemy and Haftka [6], included a vast number of approximation concepts for lower fidelity models. Early works in using variable fidelity models for optimization were largely based on heuristics, and the process was not guaranteed to converge to the high fidelity solution. Consequently, some cases converged to the low fidelity solution. New methods, some of which used gradient information, were developed and proven to converge to the high-fidelity solution after realizing that significant differences could arise in the fidelity models.

An extension to the variable fidelity optimization approach is the introduction of a scaled low fidelity approximation of the high fidelity model. Scaling based frameworks for variable fidelity optimization were first developed by Chang and Haftka *et al* [7]. They used a multiplicative scaling function to update the value of lower fidelity models to match the higher fidelity models. Then, a model management framework for managing approximation models in optimization was developed by Alexandrov *et al* [1, 2, 3, 4]. This framework uses an approximation and model management optimization (AMMO) method that incorporated first order scaling into a provably convergent methodology. This methodology could be used in various existing optimization routines such as the augmented Lagrangian method (ALM), sequential quadratic

programming (SQP), or algorithms designed to take into account the coupling of disciplines in multidisciplinary problems. Alexandrov *et al* [5] applied the framework to variable–fidelity physics models.

More recently, improvements have been made to the variable fidelity method which include making small changes in the scaling method [9] and using global scaling methods like kriging to approximate the low fidelity model to the high fidelity model [8].

1.4 Objectives

The goal of this project is to develop a variable complexity algorithm for design optimization problems. This method will be used in aerodynamic shape optimization process and will be integrated in a multidisciplinary framework. This method will take an initial approximation, optimize it and perform an accurate optimization over the initial optimum to get a better optimal solution, thereby increasing the efficiency. The project begins with an investigation of the existing variable complexity algorithms. The next step involves the testing of the algorithm on a sample problem. The project goal is to produce an algorithm that is reliable, efficient and that yields accurate optimum solutions.

To be precise, one of the aims of the project is to implement the existing the variable fidelity methods and use global scaling methods in the variable fidelity framework to improve the efficiency of the optimization process. Another aspect of the project is to explore the variable parameterization of the optimization problem. Intuitively, the method should succeed in achieving the improvement in computational time. However, the method should be examined carefully before we can comment on the convergence properties.

Then we plan to apply these methods to an aerodynamic optimization problem and verify if these produce good results.

Chapter 2

Variable Complexity Methods

2.1 Overview

Variable complexity optimization strategies can be divided into methods that use variable fidelity of the analysis and methods that use variable parameterization of the problem as explained in section 1.2. The combined usage of both high-fidelity and low-fidelity models is known as variable fidelity optimization while the changing of design variables during the course of the design process is referred to as variable parameterization method. The methods are explained in the following sections.

2.2 Variable Fidelity Framework

Variable fidelity framework is based on the trust region idea from non-linear programming. The approach inherits the mathematical robustness and convergence properties of the classical trust methods.

A simplified explanation of the framework is given below. Let $x = (x^1, x^2, ..., x^n)$ denote the design variables and suppose that there are two models, one a high physical fidelity model but with high computational cost and other, an approximate model of lower physical fidelity but with lower computational cost and let their function values be $f(x)$ and $af(x)$ and their sensitivities be $g(x)$ and $ag(x)$.

The conceptual scheme for using approximate models in optimization is shown in Figure 2.2.

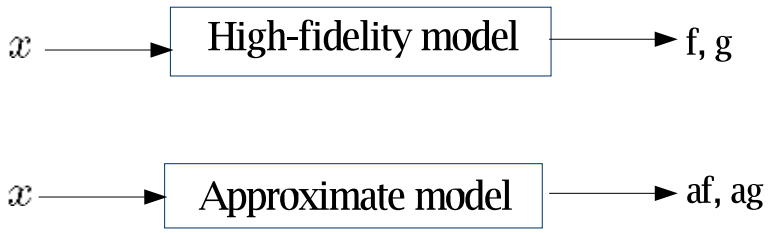

Figure 2.1: High and low fidelity models

Initially, the design generated using a model of lower fidelity but lower computational cost is checked by using information from the high-fidelity model. Again a number of optimization iterations are performed using this cheaper approximation model. At the end of optimization, the lower fidelity model is calibrated with the high fidelity model and then optimization is continued using the lower fidelity model.

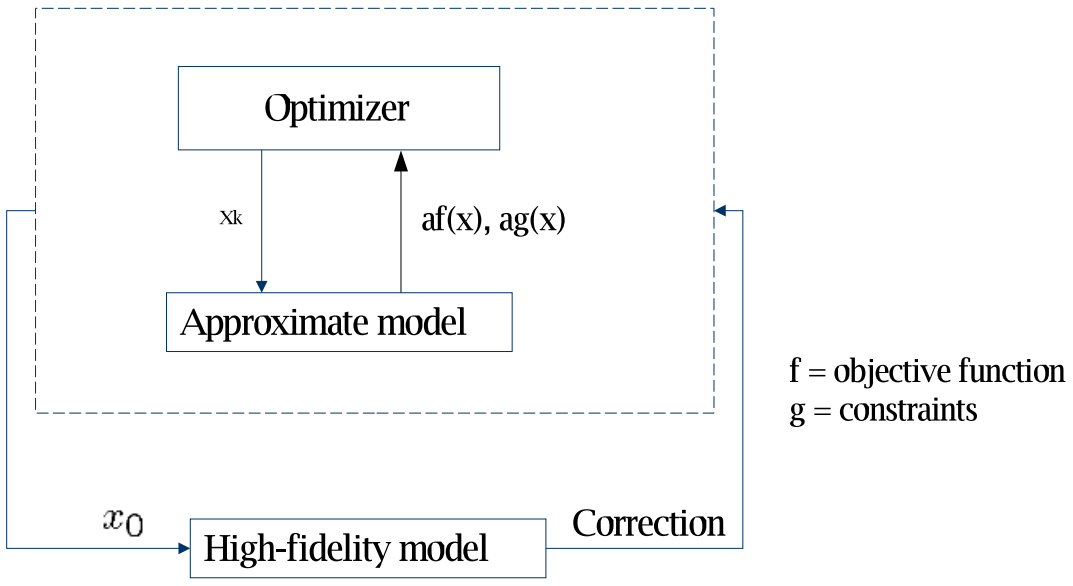

Figure 2.2: Simple description of variable fidelity framework

The trust region mechanism gives a dynamic and systematic response to poor and incorrect prediction by the approximate model. But it is not conservative to limit the progress when

the approximate model does a good job of predicting the improvement in high-fidelity model. And, the trust region gives a measure of how well the model is predicting the improvement and suggests the criteria for updating the model. Mathematically, the trust region framework is a straightforward extension of classical trust region theory.

In Section 2.3 the relevant features of the classical trust region algorithms are explained. In Section 2.4 the classical trust region approach is applied to manage the use of more general approximations. In these sections we discuss how it is decided when it might be appropriate to update the model to improve the progress of optimization based on the information used in the course of classical trust region approach. Later, in section 2.5 the implementation of this method is explained. Then, in section 2.6 a few examples of how the approximate models are matched with the high-fidelity models.

2.3 Classical Trust Region Approach

The classical trust region idea is solving a series of optimization problems using quadratic model of the original function as objective. The approach regulates the length of the steps in an iterative optimization process based on how well the current quadratic Taylor series model of f is found to predict improvement in f. This leads to an adaptive method for adjusting the size of the steps taken based on how well the local quadratic models are predicting the decrease in f.

At iteration n, a quadratic model q_n of the objective function f is created.

$$f(x_n + s) \simeq q_n(x_n + s) = f(x_n) + g_n^T s + \frac{1}{2} s^T B_n s \tag{2.1}$$

The trust region proceeds by building and minimizing quadratic models of the form Equation 2.1. However, such a model being quadratic is a good approximation only in the neighborhood of x_n. So, we restrict the step during optimization to a region in which we trust the quadratic model to approximate the function well, thus the name "trust region". So the trust region subproblem can be written by adding a constraint on the length of the step allowed as

in Equations (2.2),(2.3).

$$\text{minimize} \qquad q_n(x_n + s) \tag{2.2}$$

$$\text{subject to} \qquad ||s|| \leq \delta_n \tag{2.3}$$

So, the prospective step is accepted if its within the trust region and the move limit (δ_n) is known as trust region radius.

$$x_{n+1} = \begin{cases} x_n + s & \text{if } f(x_n + s) \leq f(x_n) \\ x_n & \text{otherwise} \end{cases} \tag{2.4}$$

In particular, after each optimization, the trust region is updated in an adaptive way based on predictive quality of the quadratic model. For example, if the model did a very good job of predicting the actual improvement of f, then increase δ_n and allow a longer step at the next optimization iteration $n+1$. However, if the model did a bad job of predicting the improvement in f, either because the function actually increased with the step s or because the function did decrease but not nearly as much as predicted by the quadratic model, then the size of the trust region used in the next optimization is decreased. The region being that quadratic model is a good approximation if its sufficiently close to x_n. Finally, if the model did an acceptable job but not a great job the trust region is left alone.

Numerically, positive constants $r_1 \leq r_2 \leq 1$ and $c_1 \leq 1, c_2 \geq 1$ are chosen that regulate the expansion and contraction of the trust region. The actual and predicted decrease is compared

$$r = \frac{f(x_n) - f(x_n + s)}{f(x_n) - q_n(x_n + s)} \tag{2.5}$$

and the trust region is updated as follows:

$$\delta_{n+1} = \begin{cases} c_1||s|| & \text{if } r \leq r_1 \\ \min\{c_2||\delta_n||, \Delta^*\} & \text{if } r \geq r_2 \\ ||s|| & \text{otherwise} \end{cases} \tag{2.6}$$

where Δ^* is an upper bound on the trust region radius. Typically, the values of r_1 and r_2 are 0.10 and 0.75 respectively.

2.4 Trust Region Method with General Approximation Models

The approximate models used in engineering practice can produce better matching with high-fidelity than quadratic model over a larger neighborhood. These approximations are based on the knowledge of the problem and thus they are specific to the application while quadratic models are always applicable.

The requirements on the approximation model at each optimization iteration:

$$a_n(x_n) = f(x_n) \tag{2.7}$$

$$\nabla a_n(x_n) = \nabla f(x_n) \tag{2.8}$$

If the model and its gradient agree with those of the actual function, it is a first-order model. If, in addition

$$\nabla^2 a_n(x_n) = \nabla^2 f(x_n) \tag{2.9}$$

then the approximation is a second-order model.

The conditions in Equations (2.7),(2.8) guarantee that the approximate model a_n is a good model of f when sufficiently close to x_n. The mechanism to regulate the use of approximation a in optimization is same as in that of the classical trust region approach. If the approximate model a is not a good predictor of the improvement of f for a longer step, the radius δ is decreased, the region in which a is an increasingly good model of f. On the other hand, if a is doing a good job of approximating the behavior of f, we do not need to decrease the length of the step, thereby avoiding the unnecessary restriction of the progress of the optimization.

The algorithm for the unconstrained minimization using general first-order approximation models is given as follows.

For n = 0,1,...until convergence do {

Choose a_n that satisfies $a_n(x_n) = f(x_n)$ and $\nabla a_n(x_n) = \nabla f(x_n)$

Find an approximate solution s_n to the subproblem

minimize $a_n(x_n + s)$

subject to $||s|| \leq \Delta_n$

Compare the actual and predicted decrease in f

$r = \frac{f(x_n)-f(x_n+s)}{f(x_n)-q_n(x_n+s)}$

Update x_n according to Equation and Δ_n according to Equation

}

2.5 Variable Fidelity Framework

The general framework for variable fidelity optimization is as shown in Figure 2.3 and is based on the work done by Alexandrov [4, 2]. The following process describes the steps of the framework:

1. *Initialization*: At the starting design point x_0, the objective and constraints are evaluated using both the high and low fidelity models.

2. *Gradient evaluation and construction of the scaling model*: At the current design point, x_n, the gradient of the objective is evaluated using both high and low fidelity models. A scaling model is constructed to ensure matching between the high and low fidelity models. Additive and multiplicative are the most common scaling methods and both can be modeled as first, second or higher order.

3. *Optimization using the low fidelity model*: The low fidelity model is used exclusively to evaluate the objective and constraints when solving the optimization problem.

4. *Evaluation of the new design and trust region management*: The high-fidelity objective function is evaluated, at the design point given by the previous step. In order to guarantee convergence of the variable-fidelity optimization framework, a trust region model management strategy is employed. A trust region ratio allows the trust region model management framework to monitor how well the approximation matches the high fidelity design space. After each completed optimization on the scaled low fidelity model, a new candidate point, x_n^* is found. A trust region ratio, ρ_n, is calculated at this new point by

$$\rho_n = \frac{P(x_n)_{high} - P(x_n^*)_{high}}{P(x_n)_{scaled} - P(x_n^*)_{scaled}}, \tag{2.10}$$

This is the ratio of actual change in the function to the predicted change of the function by the scaled lower fidelity model. Physically, ρ_n represents how good of an approximation the scaled low fidelity model is compared to the high fidelity model. If ρ_n is near 1, the approximation is quite good. If ρ_n is close to zero, the approximation is not as good, but still captures the minimization trend. If ρ_n is negative, then the point is a worse design. In this case the point is rejected, the trust region size is reduced. As long as $\rho_n > 0$, the point is accepted and the algorithm proceeds.

5. *Convergence test*: The current design point is tested for convergence and if all the requirements are satisfied, the algorithm accepts the point as optimum. Otherwise, we return to step 2 for another cycle.

2.6 Scaling Models

Scaling models can be broadly classified into two kinds, local scaling methods and global scaling methods. Most variable fidelity frameworks are based on local scaling functions, which are of two kinds: multiplicative and additive. A brief description of these methods is given in this section.

2.6.1 Multiplicative Scaling

Given the high and low fidelity models, $f_{high}(x)$ and $f_{low}(x)$, the low fidelity model can be matched to the high fidelity model by multiplying the low fidelity model with an unknown function $\beta(x)$. This can be written as

$$f_{high}(x) = \beta(x) f_{low}(x). \tag{2.11}$$

For the multiplicative scaling method, we can construct different scaling models. For example, first order method or a higher order method can be constructed. In this section, first and second order methods are presented.

The first order multiplicative approximation model is found using the scaling function $\beta(x)$ given by Chang [7]. This model ensures that the scaled low fidelity model has the same function

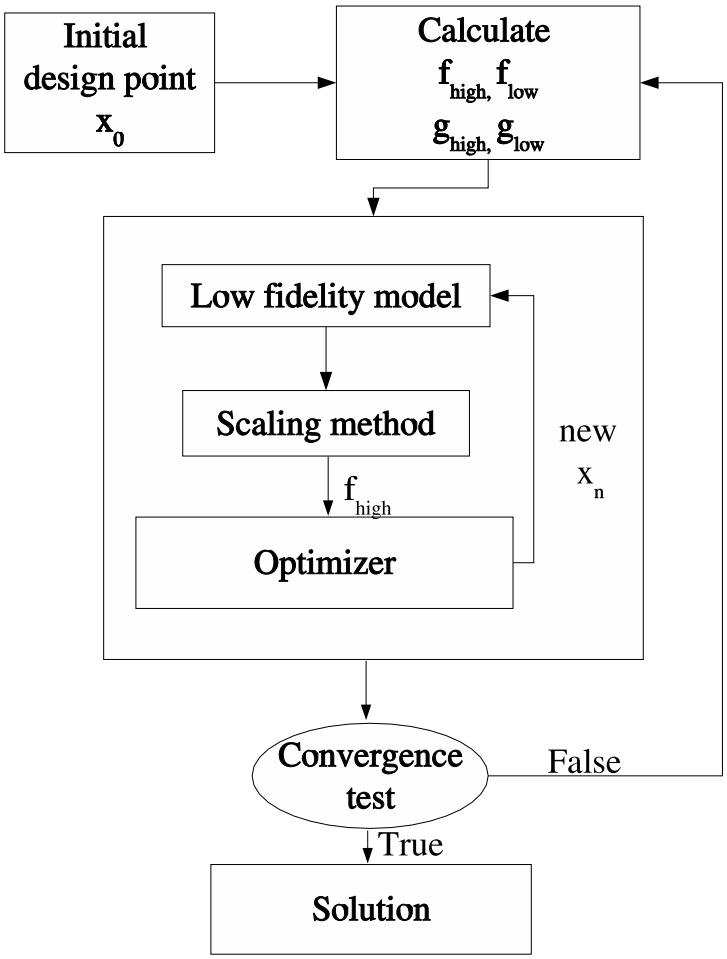

Figure 2.3: Variable fidelity framework flowchart

value and gradient as the high fidelity model, at the initial design point. At a given design point, x_n, the function is defined as

$$\beta(x_n) = \frac{f_{high}(x_n)}{f_{low}(x_n)}. \tag{2.12}$$

The scaling factor at any other point can be approximated using a Taylor series to first order:

$$\beta(x) = \beta(x_n) + \nabla\beta(x_n)^T(x - x_n). \tag{2.13}$$

Gradient information is needed to evaluate this and can be obtained by differentiating Equa-

tion (2.11).

$$\nabla\beta = \frac{\nabla f_{low}(x_n) f_{high}(x_n) - f_{low}(x_n)\nabla f_{high}(x_n)}{(f_{low}(x_n))^2}. \tag{2.14}$$

So, the low fidelity model can be matched to the high fidelity model within first order as

$$f_{high} = \beta(x) f_{low} \tag{2.15}$$

Second order multiplicative scaling is analogous to the first order method except that the Taylor series approximation is expanded to include the second order terms. The resulting scaling function is given by

$$\beta(x) = \beta(x_n) + \nabla\beta(x_n)^T(x - x_n) + \frac{1}{2}(x - x_n)^T\nabla^2\beta(x_n)(x - x_n) \tag{2.16}$$

Gradient can be calculated as before in Equation (2.14), so the remaining term is Hessian of β, which is calculated by differentiating the gradient again. Computing symmetric full-rank Hessian matrices like these would be really expensive, so approximate second order scaling is used which approximates second order information from first order information. The approximations for the Hessian are calculated using methods like Broyden–Fletcher–Goldfarb–Shanno (BFGS) and symmetric-rank-1(SR1).

2.6.2 Multiplicative Scaling with a Constant to prevent Ill-Conditioning

Multiplicative scaling method has some difficulties due to the conditioning of the beta correction when the low fidelity function approaches zero. To counter that, a small constant K is added to prevent ill-conditioning. Thus, the function is defined as

$$\beta(x_n) = \frac{f_{high}(x_n) + K}{f_{low}(x_n) + K}. \tag{2.17}$$

and the scaled function can be written as

$$a(x_n) = \beta(x)(f_{low}(x) + K) - K. \tag{2.18}$$

The gradients and the objective function values are calculated as before in Equation (2.14) but with an added constant.

$$\nabla\beta = \frac{\nabla f_{low}(x_n) f_{high}(x_n) - f_{low}(x_n)\nabla f_{high}(x_n)}{(f_{low}(x_n) + K)^2}. \tag{2.19}$$

Again, the low fidelity model can be matched to the high fidelity model within first order as

$$f_{high} = \beta(x) f_{low} \tag{2.20}$$

The value of the constant K should be large enough to prevent ill-conditioning but not be too high enough to affect the optimization process significantly.

2.6.3 Additive Scaling

A given set of high and low fidelity models, $f_{high}(x)$ and $f_{low}(x)$, can also be matched by adding an unknown function, $\gamma(x)$, to the low fidelity result. This is expressed mathematically as

$$f_{high}(x) = f_{low}(x) + \gamma(x) \tag{2.21}$$

The additive scaling function can found for by subtracting the low-fidelity function from both sides, yielding

$$\gamma(x) = f_{high}(x) - f_{low}(x) \tag{2.22}$$

Again, we can construct different scaling models, first order or a higher order. In this section, first and second order methods are presented.

The First order additive scaling method is similar to the first order multiplicative scaling method because it tries to approximate the high fidelity model to the first order by applying a correction to the lower-fidelity model.

At a given design point, the additive scaling function is given by Equation (2.22)

The additive scaling factor at any other point can also be approximated using the Taylor series of first order:

$$\gamma(x_n) = \gamma(x_n) + \nabla\gamma(x_n)^T (x - x_n) \tag{2.23}$$

Evaluating this requires gradient information which can be obtained by differentiating Equation (2.22) as shown below

$$\nabla\gamma(x_n) = \nabla f_{high}(x_n) - \nabla f_{low}(x_n) \tag{2.24}$$

Second order scaling for the additive method is as shown in the expansion

$$\gamma(x) = \gamma(x_n) + \nabla\gamma(x_n)^T(x - x_n) + \frac{1}{2}(x - x_n)^T\nabla^2\gamma(x_n)(x - x_n) \tag{2.25}$$

The gradient of γ is the same as for first order scaling. The remaining information needed is the Hessian of γ, which is given by

$$\nabla^2\gamma(x_n) = \nabla^2 f_{high}(x_n) - \nabla^2 f_{low}(x_n) \tag{2.26}$$

Computing all the terms of the Hessian is, again, expensive. Therefore, methods that approximate second order information from the gradients, such as BFGS and SR-1 are used.

2.7 Variable Parameterization

Design variables are changed during the course of the optimization process. We start the optimization with fewer design variables and then perform an accurate optimization with more design variables.

2.7.1 Overview

For the variable parameterization case, the accuracy of the problem was varied during the optimization process, starting with low number of design variables, which means less accuracy but progressively increasing the number of design variables, there by increasing the efficiency. This method is analogous to the multigrid methods in CFD.

This method is more relevant for the problems which have a smooth variation of design variables, i.e., when the objective function is smooth with respect to the design variables. This method works well with problems like shape optimization where the optimum is similar for different sets of design variables, if they have the same physical meaning. For example, in case of an airfoil optimization (a problem where the design variables control the shape) the optimum (airfoil shape which satisfies the target criteria) would be similar for different sets of design variables, say for 10 and 15 design variables which control the shape and which are equally distributed over the airfoil. The design variables can then be changed during the course of the design process.

We believe that variable parameterization is relevant in case of aerodynamic optimization problems like airfoil optimization, rotor-craft optimization.

2.7.2 Variable Parameterization in Aerodynamic Optimization

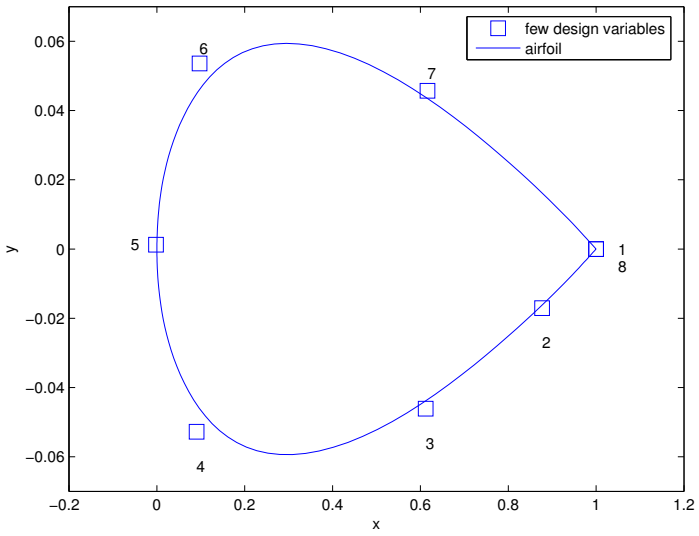

Figure 2.4: Aerodynamic optimization with few design variables

For example, we can start off with a few design variables as shown in Figure 2.4, get an idea of the problem and use engineering intuition to solve the optimization problem using the optimum got from few design variables as initial approximate for the problem with more design variables (shown in Figure 2.5).

2.7.3 Variable Parameterization in Optimal Control

It can also be used for optimal control or trajectory optimization problems as shown in Figures (2.6, 2.7). Here, the figures show the optimum of discretized brachistochrone problem with different number of design variables. Again, we can start of with lesser amount of discretization (say 10 grid points) and use that optimum as initial estimate for the problem with more number of design variables (say 50-100 points).

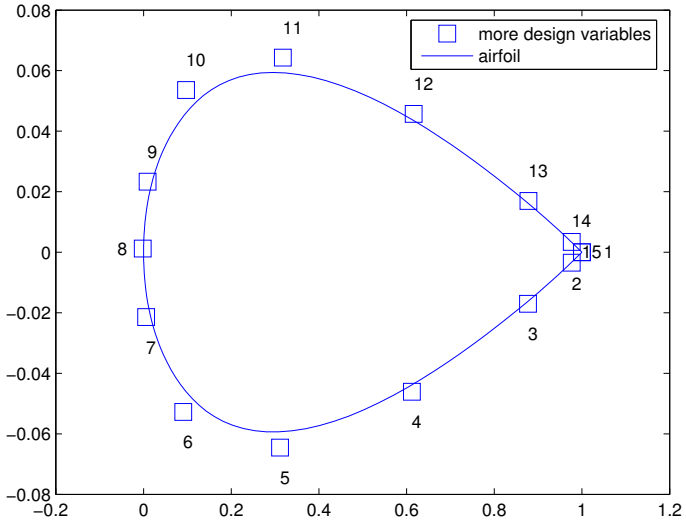

Figure 2.5: Aerodynamic optimization with more design variables

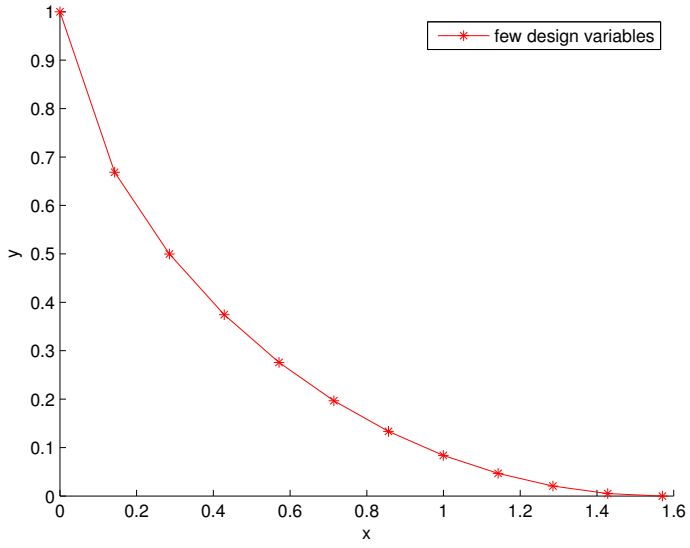

Figure 2.6: Brachistochrone problem with few design variables

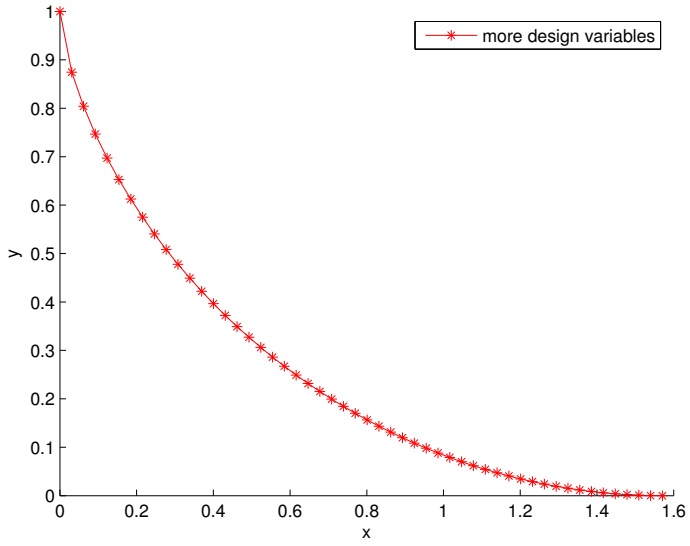

Figure 2.7: Brachistochrone problem with more design variables

2.7.4 Theory

However, there is no formal theory or mathematical proof for this method in the literature. Though the method seems intuitive, the convergence properties and the efficiency of method need to be examined.

Chapter 3

Analytical Case : The Brachistochrone Problem

3.1 Description

Much of the fundamental theory in the branch of mathematics known as calculus of variations is attributable to the Bernoulli brothers, John and James, who were friendly rivals. They would design new mathematical problems to stimulate each other in the form of challenges, one of which was the *brachistochrone* problem. The physical problem can be described as follows [13]: "Among all smooth curves joining two given points, find the one along which a bead might slide, subject only to the force of gravity, in the shortest time".

Mathematically, it can be written in the form of an optimization problem as

$$\begin{aligned} \text{minimize} \quad & T(y) && (3.1) \\ \text{w.r.t} \quad & y(x) \\ \text{s.t.} \quad & y(x_A) = y_A && (3.2) \\ & y(x_B) = y_B \\ & y = \sqrt{2gx} && (3.3) \end{aligned}$$

where T is the time taken for the bead to slide along the path given by y from (x_A, y_A) to (x_B, y_B) and g is the acceleration due to gravity.

3.2 Analytical Solution

3.2.1 General Solution

The analytical solution to the brachistochrone problem can be described as an arc of the cycloid [13]. A cycloid is a trajectory whose abscissa x and the ordinate y can be written in parametric form as

$$x^* = a(\phi - \sin\phi) \tag{3.4}$$

$$y^* = a(1 + \cos\phi) \tag{3.5}$$

3.2.2 Normalized Solution

We restrict the attention to a normalized problem. The normalization is such that we restrict the range x and domain y of the function describing the path traveled by the bead such that

$$x \in [0, \pi/2], \tag{3.6}$$

$$y \in [0, 1]. \tag{3.7}$$

for the values

$$a = \frac{1}{2} \qquad \phi = [0, \frac{\pi}{2}] \tag{3.8}$$

The continuous optimal solution to the brachistochrone problem is the curve y that minimizes the total time, which is given by the following integral:

$$T(y) = \frac{1}{2k}\int_0^{\frac{\pi}{2}} \sqrt{\frac{1+y'^2}{1-y}}dx, \tag{3.9}$$

where

$$k = \sqrt{1 + \frac{\pi}{2}^2}, \tag{3.10}$$

$$y' = \frac{dy}{dx}. \tag{3.11}$$

The optimum of this normalized problem is found by calculating the time for the cycloid trajectory described by the parametric Equations (3.4, 3.5) over one half-cycle; for a and ϕ stated as above

$$T_{min} = T^*(y^*) = \frac{\pi}{2k} = 0.843563. \tag{3.12}$$

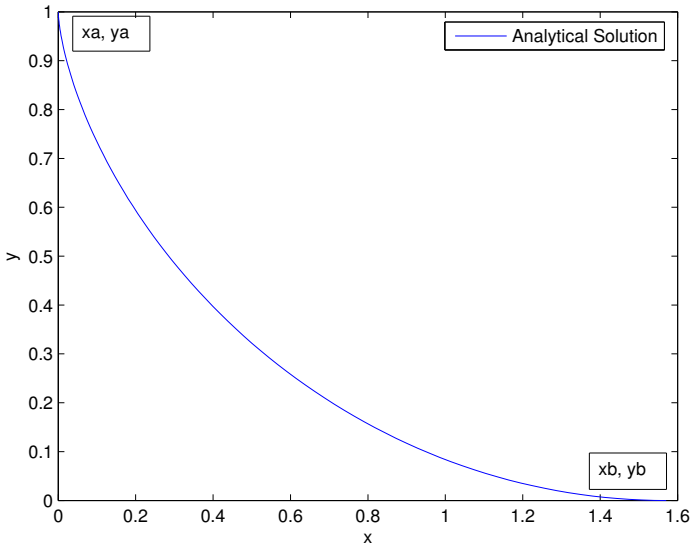

Figure 3.1: Analytical solution for the brachistochrone problem

The solution is as shown in Figure 3.1.

3.3 Discrete Solution

We considered three types of discretization: linear interpolation, polynomial and spline approximation. The discretization which we considered initially is linear interpolation which is explained in the next subsection. But, the fidelity and parameterization are the same in this case. So, In order to vary both (parameterization and fidelity), polynomial and spline interpolations are considered. These are explained in the subsequent subsections.

3.3.1 Discrete Solution using Linear Interpolation

The brachistochrone problem is solved numerically by using linearly interpolated curve $y(x)$. The path is now approximated by a vector instead of the continuous function as shown below

$$x_j \in [0, \phi/2] \qquad x_1 = 0, x_N = \frac{\pi}{2} \tag{3.13}$$

$$y_j \in [0, 1] \qquad y_1 = 1, y_N = 0 \tag{3.14}$$

The time given by (3.9) can be computed numerically using finite differencing.

$$T_D(y) = \frac{1}{2k}\int_0^{\frac{\pi}{2}} \sqrt{\frac{1+\left(\frac{y_{j+1}-y_j}{\Delta x}\right)^2}{1-\left(\frac{y_{j+1}+y_j}{2}\right)}}dx \tag{3.15}$$

Discretizing the above equation, we get

$$T_D(y) = \frac{1}{2k}\sum_{i=1}^{N-1}\sqrt{1+\left(\frac{\Delta x_j}{y_{j+1}-y_j}\right)^2}\left(\sqrt{1-y_{j+1}}-\sqrt{1-y_j}\right) \tag{3.16}$$

$$\Delta x_j = x(j+1)-x(j) \tag{3.17}$$

We minimized the time calculated using the above formula with respect to the vector of design variables y. The gradients are calculated using analytical method. The gradients are as follows

$$\nabla T_D(y)_j = \frac{1}{k}\left(\alpha(j)-\alpha(j-1)+\frac{\beta(j)-\beta(j-1)}{2\sqrt{1-y_j}}\right) \tag{3.18}$$

where

$$\alpha(j) = \frac{\Delta x_j^2(\sqrt{1-y(j)}-\sqrt{1-y(j+1)})}{\beta(j)^2(y_j-y_{j+1})^3} \tag{3.19}$$

$$\beta(j) = \sqrt{1+\left(\frac{\Delta x_j}{y_{j+1}-y_j}\right)^2} \tag{3.20}$$

The accuracy of all the results are monitored with the root-mean-square difference Y_{ERR} of the optimized discrete trajectory and the exact solution and the difference in time calculated using the optimized trajectory and the exact one, t_{err}.

$$Y_{ERR} = \sqrt{\frac{\sum_{j=1}^{N}(y_j-y_j^*)^2}{N}} \tag{3.21}$$

$$t_{err} = t_{calc} - t_{exact} \tag{3.22}$$

where y_j is the optimized discrete trajectory and y_j^* is that of the exact solution and t_{calc} is the time for the optimized discrete trajectory and t_{exact} is the time calculated for the exact solution.

The optimization was performed using Matlab optimization toolbox and the gradients have been calculated using complex-step method. No constraints have been imposed. The design variables are bounded between 0 and 1. The tolerance has been set to 10^{-6}.

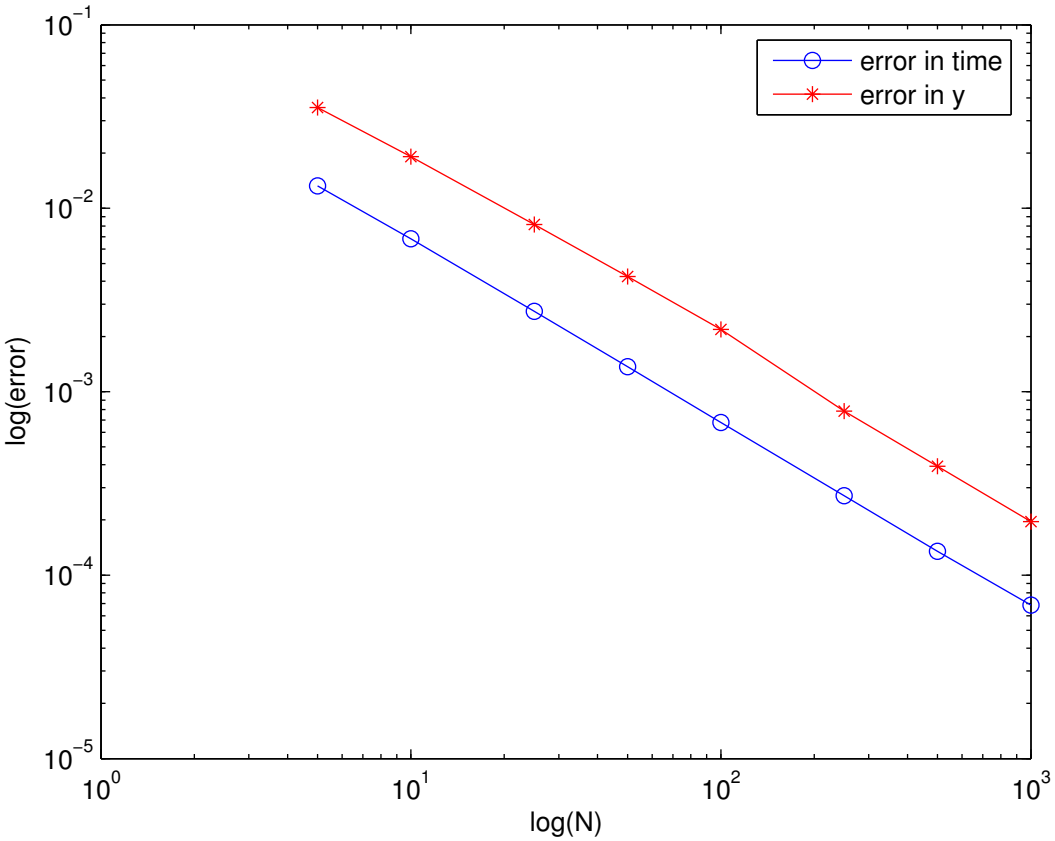

Figure 3.2: Error vs. the number of grid points for original discretization

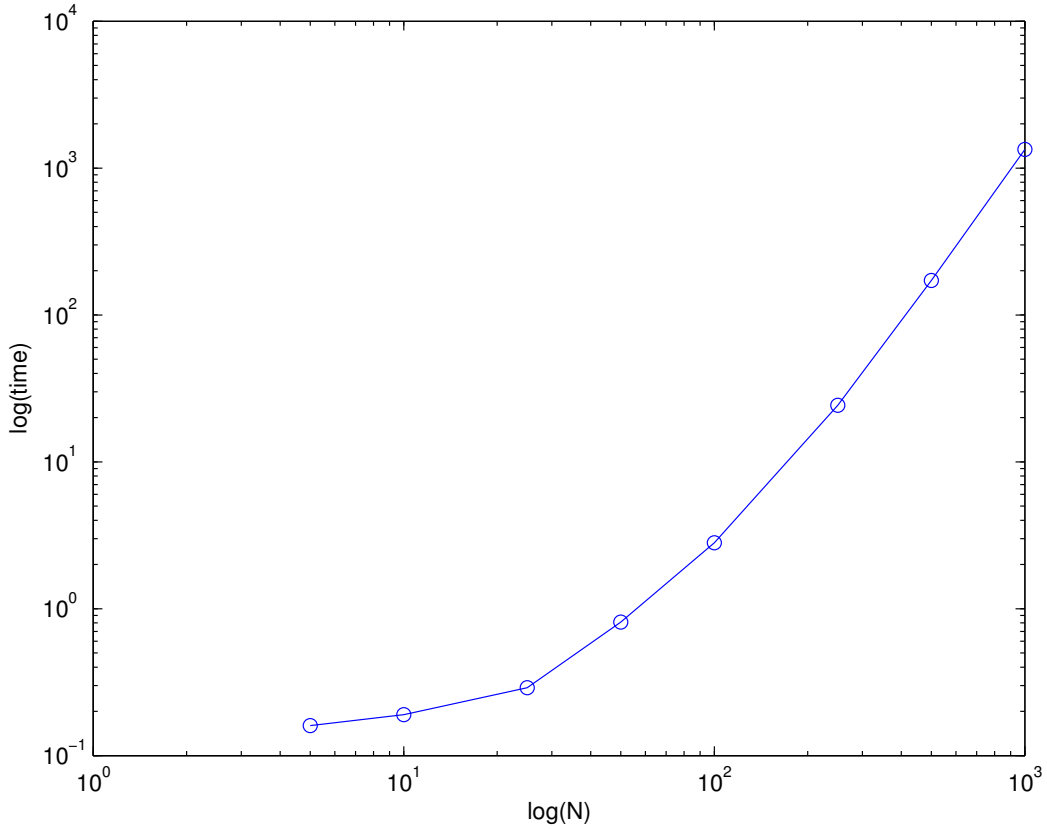

Figure 3.3: Computational time vs. the number of grid points for original discretization

The results are shown in the Figure 3.2. The figure shows the decrease in error t_{err} and Y_{err} with the number of grid points. And as mentioned before, though the number of iterations increased as the number of grid points were increased the accuracy also is better. We can see that the decrease is of second-order, which is as expected since we used central difference method for discretization. The problem in this investigation, though small, with an increase in the number of grid points there is a significant increase in the computational time taken as can be seen in Figure 3.3. Here, the increase is approximately of third-order in the logarithmic scale for the latter half of the curve. That is because the time is calculated manually and might not be accurate enough for the initial cases where the time is of the order of fraction of a second. The computational times recorded for the other cases are of the order of a few seconds so they are much more reliable. So we can safely say that, for original discretization, the decrease in error is of the order N^2 while the increase in computational time is of the order N^3 where N is the number of grid points.

3.3.2 Discrete Solution using Polynomial Interpolation

This problem is similar to the previous one, but now the curve $y(x)$ is approximated as a polynomial of specific order. The design variables are now the coefficients of the polynomial, so for a polynomial of order n, the number of design variables is $n+1$. The optimization problem can be written as

$$\begin{aligned} \text{minimize} \quad & T(y(a)) && (3.23)\\ \text{w.r.t.} \quad & a_0, a_1, \ldots, a_n \\ \text{s.t.} \quad & x_i \in [0, \phi/2] \qquad x_1 = 0, x_N = \tfrac{\pi}{2} && (3.24)\\ & y_i \in [0, 1] \qquad y_1 = 1, y_N = 0 && (3.25) \end{aligned}$$

where

$$y_i = a_0 x_i^n + a_1 x_i^{n-1} + a_2 x_i^{n-2} + \ldots + a_{n-1} x_i + a_n \tag{3.26}$$

The calculation of time is same as the discretized integral (3.16) with the ordinate y approximated as a polynomial of order n with x_i as shown in Equation (3.26). The higher the order of the polynomial, the more accurate the result.

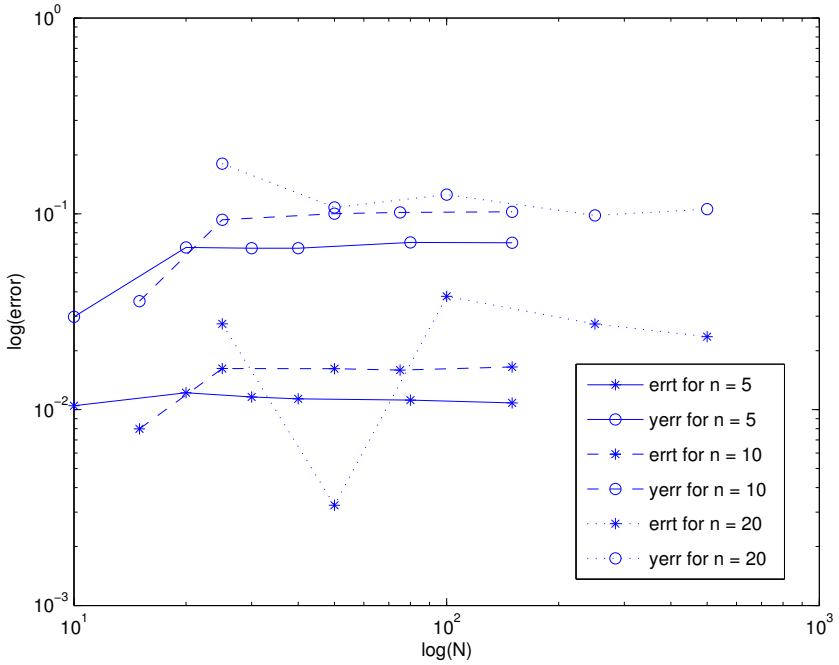

Figure 3.4: Error vs. the number of grid points for the polynomial approximation

The results are shown in Figure 3.4. The figure shows the decrease in error t_{err} with the number of grid points for polynomials with different orders. We expected that the error t_{err} can be decreased with increasing order of the polynomial or by increasing the number of grid points. Even though some of the results followed the trend explained above, the results were not entirely satisfactory. For some cases, the error actually increased with increasing the number of grid points as can be observed for cases $n = 5$ and $n = 10$. This can be explained by the fact that the polynomial is a bad approximation of the curve.

3.3.3 Discrete Solution using Spline Interpolation

In this problem, we approximate the curve as a cubic spline and the design variables are the ordinates of the spline control points. A cubic spline is a piecewise cubic polynomial such that the function, its derivative and its second derivative are continuous at the control points. Consider the cubic spline for a set of N_C points $(Y_1, Y_2, ..., Y_{N_C})$, the optimization problem can

be written as

$$\text{minimize} \qquad T(y(Y))) \tag{3.27}$$

$$\text{w.r.t.} \qquad Y_1, Y_2, \ldots, Y_{N_C}$$

$$\text{s.t.} \quad x_i \in [0, \phi/2] \qquad x_1 = 0, x_{N_G} = \tfrac{\pi}{2} \tag{3.28}$$

$$y_i \in [0, 1] \qquad y_1 = 1, y_{N_G} = 0 \tag{3.29}$$

where the i^{th} piece of the spline can be represented by

$$y_i(s) = a_i + b_i s + c_i s^2 + d_i s^3 \tag{3.30}$$

where s is a parameter $s \in [0, 1]$ and $i = 0, ..., n-1$. The C_2 continuity conditions mentioned above, lead to a simple tridiagonal system that can be solved easily to give the coefficients of the polynomials. In this problem, we create a cubic spline using the control points and the calculation of time is performed the same way as in the previous cases with the cubic spline points as the grid points. The control points are used as design variables.

In this case the fidelity of the analysis, which depends on the number of grid points is separated from the optimization problem parameterization, which depends upon the number of design variables. This enables us to vary both the parameterization and fidelity of the problem. Here, the parameterization is based on the number of control points (N_C) while fidelity is based on the number of grid points (N_G).

The problem was solved for a varying number of grid points with the number of control points being the same, variable fidelity approach and in other case, the problem was solved for varying number of design variables (which are the control points of the cubic spline), variable parameterization method. The results are shown in Figure 3.5. It shows the decrease in error (t_{err}) with increasing the number of grid points for splines for different number of control points. It is easily seen that the error can be decreased by increasing the number of control points (keeping the grid points same) or by increasing the number of grid points while keeping the control points same. However, the decrease achieved by increasing the number of grid points for the same number of control points is almost negligible. But, with increasing the number of control points noticeable increase in the accuracy can be achieved as can be seen in Figure 3.5.

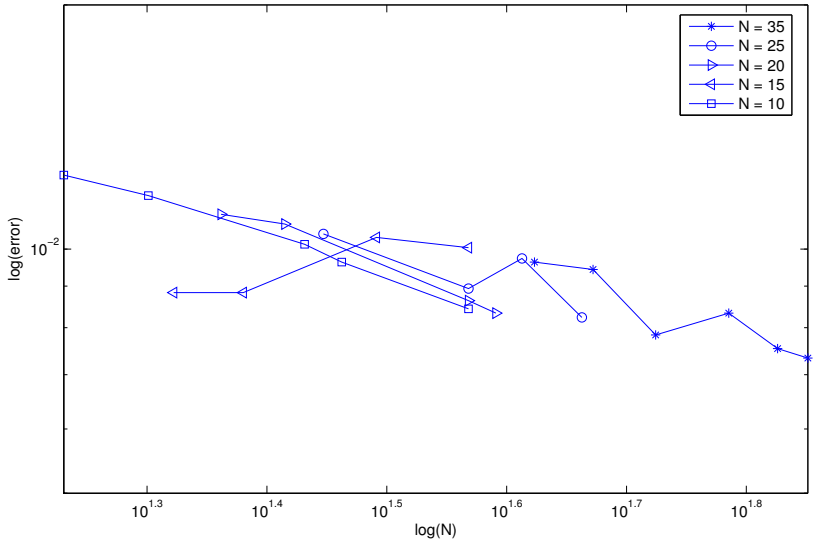

Figure 3.5: Error vs. the number of grid points for the spline approximation

3.4 Variable Complexity Optimization Experiments

In the previous section, we implemented different discretization strategies for the brachistochrone problem and explored their convergence properties with the accuracy of discretization. We tried to use insights gained in the previous section to implement the variable parameterization method and to do variable fidelity optimization.

Optimization was performed using the same Matlab optimization toolbox and the gradients were calculated using the complex-step method [10]. If F is the result of any (real-valued) numerical algorithm, and x is one of the input variables, then the derivative of F with respect to x is

$$\frac{\partial F}{\partial x} = \frac{Im[F(x+ih)]}{h} \tag{3.31}$$

where i is the imaginary unit, and h is a small step size. No constraints have been imposed. The design variables are bounded between 0 and 1. The convergence tolerance has been set to 10^{-6}.

We should keep in mind that the results show only number of iterations, while the function evaluations are $n+1$ times the number of iterations as the gradients have to be included (since

the complex step requires a function evaluation for each design variable).

This section is divided into variable parameterization and variable fidelity optimization. Each subsection gives an overview of the implementation of the method and explains the results for a few of the sample cases that have been examined. Finally, a few conclusions and suggestions are made about both the methods.

3.4.1 Variable Parameterization

For the variable parameterization case, the dimensionality of the problem was varied during the optimization process, starting with low number of design variables (spline points) and progressively increasing the number of design variables.

This method is implemented for both original discretization (discrete solution using linear interpolation as explained in section 3.3) and discrete solution using spline interpolation. We will explain about the implementation and the results of variable parameterization method on original discretization first and then we will do the same for spline interpolation.

3.4.2 Linear Interpolation

The basis for original discretization is that the brachistochrone problem is solved numerically by using a linearly interpolated curve i.e the path is approximated as a vector instead of continuous function. In simple words, the curve is approximated as a linear interpolation between a series of ordinates as shown in the figure. According to the experiments we done on the brachistochrone problem using linear interpolation, we have observed that both the computational time and the accuracy (measured in terms of t_{err} and Y_{err}) increased with the number of grid points (N).

Variable parameterization method involves changing the design variables so, here, we change the grid points during the course of the optimization process. In other words, we start off with few grid points (N_i), get an optimum and use that optimum as initial guess for the problem with more grid points(N_f). The problem with fewer grid points (N_i) is called *initial* problem while the problem with more grid points (N_f) and which uses the optimum of initial problem as initial guess is called *parameterization method* problem. The problem with more number of grid points (N_f) but with a straight line as initial guess is called *direct* problem (which is the

actual problem in consideration).

The variable parameterization method has been tried for various combinations of initial and final number of grid points. The results can be seen in Table 3.1. The method has been tried for different cases, that is, for varying degrees of accuracy for the initial problem. The table shows the results for initial problems with 5, 10, 25 and 50 grid points. For each of these, The table shows the number of grid points for the final problem (N_f), number of high fidelity function calls for the variable parameterization method and the function calls for the direct (actual) problem. The number of iterations for problem with more number of grid points (N_f) and which uses the optimum of the problem with less number of design variables (N_i) as initial guess are shown in column n_{method} and the number of iterations for the problem with the same number of grid points (N_f) but with a straight line as initial guess are shown in column n_{direct}. The number of iterations (n_i) for the problem with less number of design variables are not shown in the table, but they are shown in the top row (along with N_i), because they are same for all the problems which use the same number of initial grid points. By comparing n_i and n_{method}, we can observe the savings achieved by the variable parameterization method.

From the table, we can see that using an approximate optimum as an initial guess results in fewer iterations, saving the computational time. Since we have observed that the computational time is of the order of N^3, the savings can be calculated by considering only n_{method} and n_{direct} especially if the difference between N_i and N_f which is the case for most cases. For most of the cases, the variable parameterization method managed to reduce the function evaluations by 20–50%.

Only thing to notice is that if the difference in dimensionality between the initial problem and the actual problem is large, there might be some difficulties in the convergence and the efficiency can be lost. For example, for the case $N_i = 25$ and $N_i = 150$, the parameterization method takes 13 function evaluations which is more than the number of function evaluations taken by the direct (actual) problem (12 function evaluations). This can be explained with the ill conditioning arising due to the huge difference in dimensionality.

Table 3.1: Variable parameterization optimization results for the brachistochrone problem

$N_i = 5$, $n_i = 3$			$N_i = 10$, $n_i = 8$			$N_i = 25$, $n_i = 9$			$N_i = 50$, $n_i = 10$		
N_f	n_{method}	n_{direct}	N_f	n_{method}	n_{direct}	N_f	n_{method}	n_{direct}	N_f	n_{method}	n_{direct}
10	4	8	50	6	10	50	5	10	100	5	11
20	5	9	100	8	11	100	5	11	150	5	11
30	6	9	150	9	11	150	13	12	250	6	12

3.4.3 Spline Interpolation

But in case of the original discretization, the fidelity, which is based on the grid points and the complexity, which is based on the design variables are linked as the design variables and the grid points are the same. In order to better examine the variable parameterization method, we decided to separate the accuracy of the objective function from the design variables. The spline approximation serves this purpose. In the problem using spline approximation, the fidelity of the analysis and the complexity is independent, allowing us to examine the properties of each method separately. Here, the complexity of the problem is related to the number of control points as they determine the number of iterations taken by the optimizer while the fidelity of the problem is related to the number of grid points as they determine the computational time taken for each function evaluation.

In the application of variable parameterization method to spline interpolation we changed the design variables, N_C, during the course of the optimization process. We started of with a low number of design variables (fewer control points in the spline, N_{C_i}) and optimized the discretized problem. Then we performed an accurate optimization with more design variables (more control points, N_{C_f}) using the optimum of the previous problem as an initial guess. Again, the problem with fewer control points (N_{C_i}) is called *initial* problem while the problem with more control points (N_{C_f}) and which uses the optimum of initial problem as initial guess is called *parameterization method* problem and the problem with more number of grid points (N_{C_f}) but with a straight line as initial guess is called *direct* problem.

This method has been applied for different number of grid points. A few of the sample cases that were tried are tabulated. The Table 3.2 shows the results for problems with 30, 40 and

Table 3.2: Variable parameterization optimization results for spline interpolation

N_G	N_{C_i}	N_{C_f}	n_{low}	n_{method}	n_{direct}
20	4	6	13	24	26
30	4	6	28	36	47
50	4	6	21	31	70
30	3	7	10	33	53
50	3	7	18	57	80

50 grid points and for each case, it shows the initial and final number of design variables, the low and high fidelity function calls and the number of function calls for the actual problem. Again, as in the previous case, the columns N_{C_i} and N_{C_f} denote the number of initial and final design variables (control points) respectively. The number of function evaluations for the initial problem are shown in column n_{low} while columns n_{method} and n_{direct} denote the number of function evaluations for the variable parameterization method and the function calls for the direct (actual) problem.

The results show that a significant increase can be obtained through this variable parameterization method. All the parameters have the same or similar meaning as explained earlier. Here, even though we do not have the exact relationship of how computational time varies with number of control points, we can safely assume that time taken for initial problem is only a fraction of the time taken for the actual problem. We can see that the variable parameterization method has managed to save the number of function evaluations by 10 –57%. Though the variable parameterization works in most cases, it is not consistent. For example, the savings that the method generates does not follow a specific trend which can be observed in Table 3.2.

3.4.4 Variable Fidelity Optimization

We also implemented the variable fidelity method as explained in section 2.5. In the variable fidelity framework, the design variables are kept the same but the optimization problem is solved on a low-fidelity model (less accurate but faster) and corrections are applied using a high-fidelity model (accurate but expensive i.e takes a lot of computational time).

For the brachistochrone problem, The high-fidelity model is the same as shown in Equation (3.9). The low-fidelity model is obtained by adding a noise function (η) to the actual problem as shown in Equation (3.32).

$$\begin{aligned} \text{minimize} \quad & T(y+\eta) && (3.32) \\ \text{w.r.t} \quad & y(x) \\ \text{s.t.} \quad & y(x_A) = y_A \qquad \eta(x_A) = 0 && (3.33) \\ & y(x_B) = y_B \qquad \eta(x_B) = 0 \\ & y = \sqrt{2gx} && (3.34) \end{aligned}$$

Linear interpolation is used in discretizing the brachistochrone problem, so, the grid points are the design variables. Here, even though the low-fidelity and high-fidelity models take approximately the same computational time we use them in the variable fidelity framework. This test problem (brachistochrone problem) is only to verify that the methods converge and work properly and to compare the relative merits of each method.

The variable fidelity method was tried with both additive and multiplicative scaling methods for different cases (different grid points) and a few of them are tabulated. The Table 3.3 shows the results for problems with 7,10,15 and 25 grid points (N_G) for both additive and multiplicative scaling methods. The table shows the number of high and low fidelity function calls for both variable fidelity frameworks (additive and multiplicative) and the function calls for the actual problem. The number of function calls for the direct (actual) problem are shown in column n_{direct}. For the framework, the number of low fidelity and high Fidelity function calls are shown in columns n_{low} and n_{high} respectively and the savings it achieved compared to the actual problem is shown in column *savings*.

The savings are calculated by the high fidelity function evaluations that the framework has managed to save compared to the actual number of function evaluations (for the direct problem). However, this is not entirely true because in most cases the time taken for low-fidelity model, though less compared to high-fidelity model, is not negligible. So, the real savings can be obtained by comparing the CPU time taken for both the framework and the actual problem. Here, we can not compare the actual savings in computational time because the time taken

for the low-fidelity model is the same as the high-fidelity model (as the low-fidelity model is obtained by adding noise to the actual problem). However, we can derive a simple equation to decide what fraction of time of the actual problem the low-fidelity model should take for the framework to achieve savings in computational time. The derivation is explained below.

let the average computational time taken for the low and high fidelity models be t_{low} and t_{high}. Let the number of function evaluations taken by the optimizer for the actual problem be n_{direct}. Given that the variable fidelity framework takes n_{low} low fidelity evaluations and n_{high} high fidelity evaluations, we can calculate the ratio of computational times taken by the low-fidelity and high-fidelity models for each function evaluation, using a condition necessary to achieve savings in computational time. This is explained in Equation (3.35).

$$\text{time taken for framework} \leq \text{time taken for actual problem} \tag{3.35}$$

$$\Rightarrow n_{low}t_{low} + n_{high}t_{high} \leq n_{direct}t_{high} \tag{3.36}$$

$$\Rightarrow \frac{t_{low}}{t_{high}} \leq \frac{n_{direct} - n_{high}}{n_{low}} \tag{3.37}$$

For example, in the case of brachistochrone problem with 15 grid points the framework (with additive scaling method) can produce savings if the time taken by the low-fidelity model for each function evaluation were either less than or equal to one-third of the time taken by the high-fidelity model.

However, since the brachistochrone problem was only to verify the framework, we ignored the low-fidelity model when we calculated the savings and those are the savings tabulated in Table 3.3. We can easily see that additive scaling performed better than the multiplicative scaling, though both succeeded in reducing the number of high-fidelity function calls. Additive scaling, on an average, achieved between 57–66.6% in savings while multiplicative scaling could only achieve between 33.3–43% savings in the number of high-fidelity function evaluations.

3.4.5 Conclusions

Variable parameterization results for the original discretization show a decrease in overall computational cost of 30-50%, as shown in Table 3.1. This increased efficiency is due a less costly analysis and also due to a reduction in the number of variables (because the fidelity of the

Table 3.3: Variable fidelity optimization results for the brachistochrone problem

	n_{low}	n_{high}	n_{direct}	savings	n_{low}	n_{high}	n_{direct}	savings
	$N_G = 7$				$N_G = 10$			
Direct	–	–	7	0	–	–	8	0
Additive	14	3	7	57%	15	3	8	62.5%
Multiplicative	12	4	7	43%	13	5	8	37.5%
	$N_G = 15$				$N_G = 25$			
Direct	–	–	8	0	–	–	9	0
Additive	15	3	8	62.5%	15	3	9	66.6%
Multiplicative	13	5	8	37.5%	15	6	9	33.3%

analysis is linked to the fidelity of the parameterization). To separate the effects of accuracy of discretization (variable fidelity analysis) and design variables (variable parametrization), a spline representation was chosen. The variable parameterization method succeeded in saving the computational time while varying the fidelity of the analysis could not achieve the same reduction in computational cost. As effective as the variable parameterization method was in this case, further mathematical study is necessary to determine the convergence properties or this method.

Variable fidelity methods on the other hand are guaranteed to converge and they did, as can be seen in the results in Table 3.3. Both scaling methods succeeded in reducing the number of high-fidelity function calls, but additive scaling performed better than the multiplicative scaling.

Chapter 4

Airfoil Optimization

4.1 Airfoil Optimization

The airfoil optimization was done using the two-dimensional Navier–Stokes flow solver CYCLONE and the aerodynamic shape optimization algorithm OPTIMA-2D, both developed by Nemec and Zingg [11, 12], at the University of Toronto Institute of Aerospace Studies (UTIAS).

4.2 Overview

CYCLONE solves the compressible thin-layer Navier–Stokes equations, which are discretized on structured grids for single-element airfoils. Aerodynamic shape optimization capability developed for the CYCLONE flow solver is based on an algorithm referred to as OPTIMA-2D, where a gradient-based numerical optimization approach is used to determine optimal airfoil shapes and configurations. The gradient is computed via the discrete-adjoint method. The convergence of OPTIMA-2D has been accelerated by a Newton–Krylov algorithm.

4.2.1 Problem Formulation

The aerodynamic shape optimization problem consists of determining values of design variables X, such that the objective function J is minimized

$$\begin{aligned} \text{minimize} \quad & J(X, Q) \\ \text{w.r.t.} \quad & X \\ \text{s.t.} \quad & C_j(X, Q) \leq 0 \end{aligned} \tag{4.1}$$

(4.2)

where the vector Q denotes the conservative flow variables and C_j denotes the problem constraints such as thickness constraints.

4.2.2 Design Variables

The design variables are primarily the parameters that control the shape of the airfoil. Additional design variables like the angle of attack could be chosen depending on the problem of interest.

B-splines are used to parameterize the airfoil shape. The following development, which describes the construction of a B-spline curve and the initial airfoil shape approximation, was developed by Nemec [11, 12], based on the work of Boor and Hoschek. The parametric representation of an airfoil shape with a B-spline curve is given by

$$x_a(w_j) = \sum_{i=1}^{n+1} X_i^c B_{i,k}(w_j) \tag{4.3}$$

$$y_a(w_j) = \sum_{i=1}^{n+1} Y_i^c B_{i,k}(w_j) \tag{4.4}$$

where (x_a, y_a) are the Cartesian coordinates of the airfoil surface, $B_{i,k}$ are the B-spline basis functions of order k, (X_i^c, Y_i^c) are the coordinates of the B-spline control points and the total number of control points is $n + 1$.

Before the starting the optimization process the location of the B-spline control points are chosen such that they best approximate the initial airfoil shape. Say, the initial airfoil surface is defined by a set of points $P_j = P(x_j^*, y_j^*)$, the location of the control points $D(X_i^c, Y_i^c)$ is

found such that the distance between data point P_j and the corresponding B-spline curve point $C_j = C[x_a(w_j), y_a(w_j)]$ is minimized, i.e we solve the following optimization problem:

$$\min_D \sum_{j=1}^{N} \|P_j - C_j\| \tag{4.5}$$

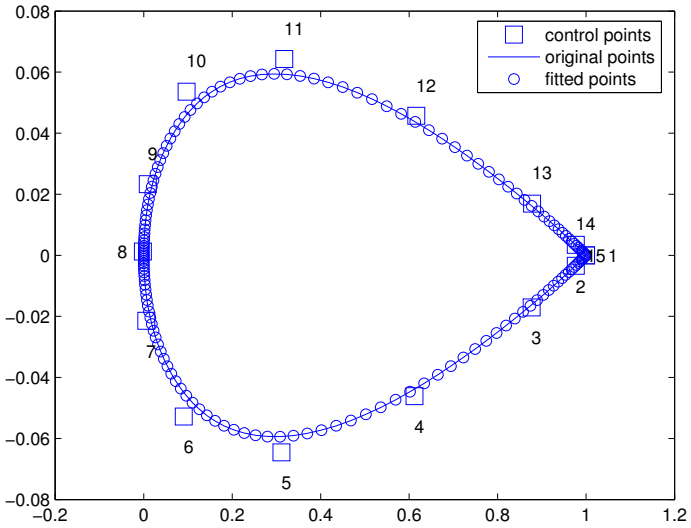

Figure 4.1: Design variables

An example is shown in Figure 4.1, where cubic B-splines constructed using 15 control points are used to approximate the NACA 0012 airfoil. Here, the points representing the initial airfoil surface are shown as *original points*. The location of the B-spline control points is determined such that the distance between *original points* and the B-spline curve points (shown as *fitted points*) is minimum. The obtained control points are shown as *control points*.

The accuracy of the B-spline curve can be improved by increasing the number of control points. Nemec's numerical experiments performed on the airfoil show that 25 control points are sufficient to reach an error tolerance of $5 \times 10^{-5}c$ (c is chord-length of the airfoil), while the manufacturing tolerance used the aircraft industry is typically assumed to be $2 \times 10^{-4}c$, which means that in practice there is no point in obtaining a better B-spline approximation of the original airfoil.

The design variables are the vertical coordinates of the B-spline control points. The control

points associated with the leading and trailing edges remain fixed. The angle of attack can also be set as a design variable.

4.2.3 Objective Functions

Lift-constrained drag minimization, lift-to-drag ratio maximization and inverse design optimization problems are considered.

For the lift-constrained drag-minimization and lift-enhancement problems, the objective function has the form

$$J = \begin{cases} w_L(1 - \frac{C_L}{C_L^*})^2 + w_D(1 - \frac{C_D}{C_D^*})^2 & \text{if } C_D > C_D^* \\ w_L(1 - \frac{C_L}{C_L^*})^2 & \text{otherwise} \end{cases} \tag{4.6}$$

where C_D^* and C_L^* represent the target drag and lift coefficients, respectively. The weights w_D and w_L are constants specified by the user. For the maximization of lift to drag ratio problem, the objective function is simply

$$J = \frac{C_D}{C_L} \tag{4.7}$$

For the inverse design problem, the objective function is given by

$$J = \frac{1}{2}\int_S (C_p - C_p^*)^2 ds \tag{4.8}$$

where C_p^* represents the target pressure distribution which is user specified and S denotes the airfoil boundary. When solving the problem, optimizer finds an airfoil whose shape matches the target pressure distribution.

4.2.4 Constraints

The constraints are primarily the airfoil thickness constraints that are necessary for feasible designs. These constraints protect the wing cross section from being thin, which causes the structure to fail. There are also constraints to ensure good grids. The leading edge radius and the trailing edge angle are also constrained for practical designs. The thickness constraints are

given by

$$h^*(x_j) - h(x_j) \leq 0 \tag{4.9}$$

where $h^*(x_j)$ represents the maximum allowable thickness at location x_j and $h(x_j)$ represents the current airfoil thickness.

4.2.5 Optimizer

The optimizer used for aerodynamic optimization is BFGS quasi-newton algorithm with a back tracking line search. The problem is cast as an unconstrained problem by using penalty method for the constraints. A brief description of the algorithm is as given below.

The goal of the optimizer is to drive the norm of gradient towards zero. A search direction, s_n is calculated at every iteration n as follows

$$s_n = -H_n G_n \tag{4.10}$$

where H is an approximation of the inverse Hessian calculated using BFGS method and G is the gradient of the objective function. Once the search direction is calculated, the updated design variables are calculated using the relation

$$x_{n+1} = x_n + \beta_n s_n \tag{4.11}$$

where β is the step length calculated using the backtracking line-search method. Once the updated design variables are determined, another flow solution is generated and the new objective and gradients are calculated, a new search direction is calculated. Again, the updated design variables are calculated and the same process is followed. The stopping criteria is the reduction of the norm of the gradient to required tolerance.

4.3 Variable Complexity Experiments

The variable complexity optimization methods that we tried can be broadly classified into methods that use variable fidelity of the model and methods that vary the parameterization of the problem, as explained in Section 1.2.

First, we will examine the variable fidelity method, that is, we use an initial approximation with a coarse grid, optimize it and validate it using an accurate analysis with a fine grid over the initial optimum. We will also examine the variable physical fidelity method, which uses different physical models for low and high fidelity evaluations. In the current work, we solve the Navier–Stokes equations with a turbulence model on a fine grid as the high-fidelity model and solve the Euler equations on a coarse grid for the low-fidelity model. We tried both additive and multiplicative scaling for all these sample cases.

We then examined the variable parameterization method by changing the design variables during the course of the optimization process as we did in the case of the brachistochrone problem. We started of with fewer design variables (fewer control points in the spline approximation), optimized the discretized problem and then performed an accurate optimization with more design variables. Optimization was performed on a fine grid with Navier–Stokes equations with turbulence model in both the cases of parameterization.

4.4 Variable Fidelity Method

4.4.1 Overview

The variable fidelity method uses the same physical models (Navier–Stokes in this case) but uses a low-fidelity model which takes less computational time to perform optimization and corrections are applied using a high-fidelity model. The low-fidelity model is less accurate compared to the high-fidelity model but when used in the framework (i.e. when validated by the high-fidelity model), it converges to the high-fidelity solution.

We considered three aerodynamic optimization problems as our test cases. The local scaling methods used are additive, multiplicative and multiplicative scaling with a constant. The variable fidelity method has been applied to lift-constrained drag minimization, lift-to-drag ratio maximization and inverse design optimization. The results are shown for these optimization problems in the following subsections.

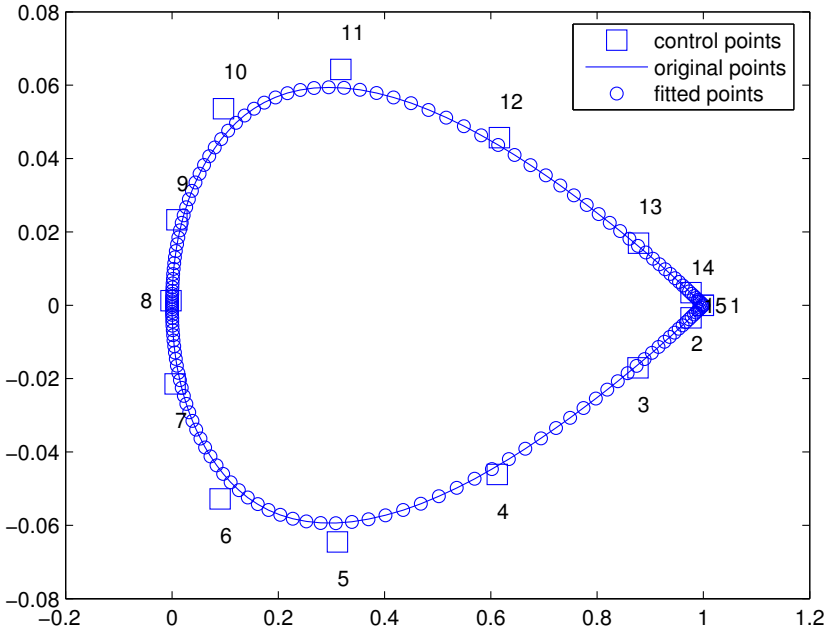

Figure 4.2: Initial airfoil geometry, NACA 0012 airfoil

4.4.2 Lift-Constrained Drag Minimization

The optimization problem considered here is transonic lift-constrained drag minimization of an airfoil. The NACA 0012 airfoil shown in Figure 4.2 is used as initial airfoil geometry. The freestream conditions are $M_\infty = 0.7$ and $Re = 9 \times 10^6$. The design variables are control points numbered 9, 10, 11 and 12 starting from the trailing edge and going clockwise. The target lift coefficient is $C_L^* = 0.4728$ while the target drag coefficient is $C_D^* = 0.0112$. The high-fidelity model here is a full Navier–Stokes solution with a turbulence model on a fine grid (283×65) and the low-fidelity model is also a Navier–Stokes solution over a coarse mesh (201×45). The flow solution takes 160 seconds for the fine grid while it takes about 30 seconds for the coarse one. The time measured here is CPU time on an AMD Athlon desktop (1 GHz).

The problem was solved with the high-fidelity model (using Navier-Stokes equations) alone using OPTIMA-2D in order to obtain a benchmark for the number of iterations or function evaluations to find the optimum. The values of C_L and C_D are 0.4851 and 0.0121 after 29 design iterations (to reduce the L_2 norm of the gradient by five orders of magnitude), which

Table 4.1: Variable fidelity optimization results for the lift constrained drag minimization problem

	Direct Optimization	Framework
C_L	0.4851	0.4843
C_D	0.0121	0.0122
High Fidelity Function Calls	29	4
Low Fidelity Function Calls	–	89
CPU time	90 min	110 min

means a 20% reduction in the drag coefficient. In terms of CPU time, the optimization process took 90 minutes.

Table 4.1 shows the results for additive scaling method. The table shows the number of high and low fidelity function calls for the variable fidelity framework and the function calls for the actual problem. The variable fidelity method managed to achieve almost the same reduction in drag coefficient (17%) as the high fidelity problem, as can be seen in the table, but could not succeed in reducing the computational time. The additive scaling method took to 110 min to converge while the direct optimization using high-fidelity took 90 min. This is due to the fact that the time taken for a low-fidelity analysis, though less than the time for a high-fidelity analysis, is not negligible and the optimization problem is not big enough to achieve significant decrease in the computational time. In order to better explore the variable fidelity framework, larger optimization problems are considered.

4.4.3 Inverse Design

The problem here is as presented in Equation (4.8). The target pressure distribution is that of a RAE 2822 airfoil configuration and the initial pressure distribution is that of the NACA 0012 airfoil, as shown in Figure 4.3. The freestream conditions are $M_\infty = 0.7$, $Re = 9 \times 10^6$ and angle of attack $\alpha = 3\,^\circ$. 12 control points are used as design variables (control points numbered 1, 8 and 15 are kept constant during the optimization). The high-fidelity model here, again, is a full Navier–Stokes solution with a turbulence model on a fine grid (283×65)

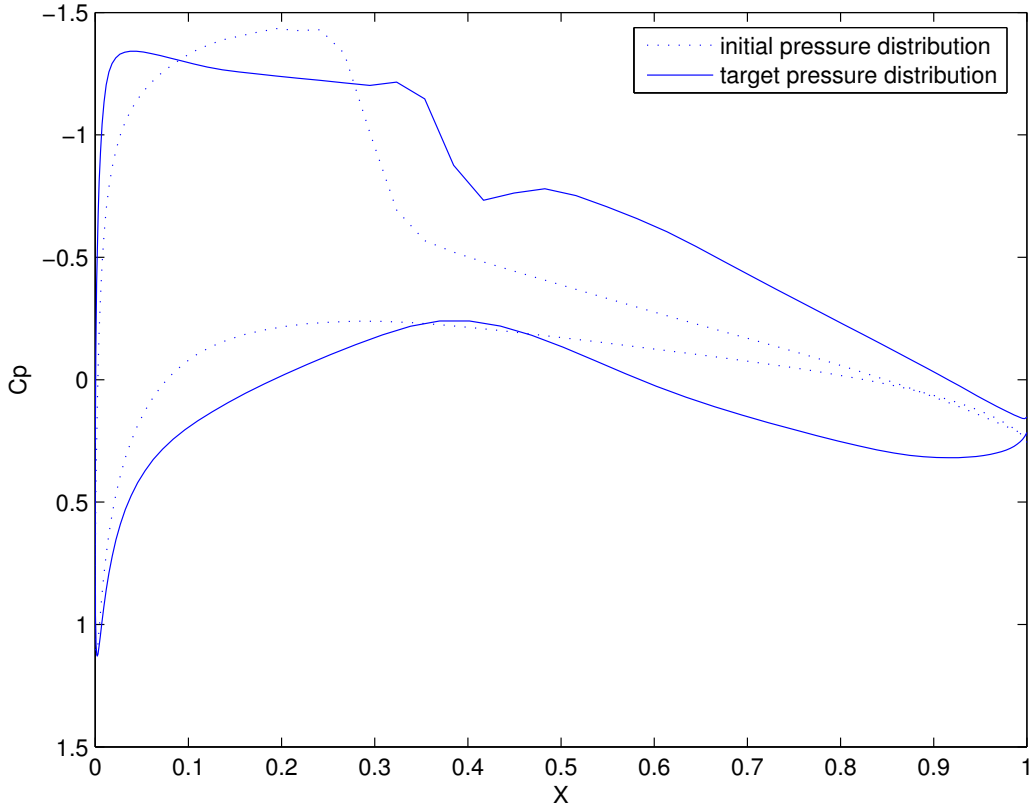

Figure 4.3: Initial and target pressure distributions for inverse design

and the low-fidelity model is also a full Navier–Stokes solution over a coarse mesh (201×45). An optimization routine (a complete flow solution and gradient analysis) takes 220 seconds for the fine grid while it takes 40 seconds for the coarser grid.

Again, in order to obtain a baseline number of iterations or function evaluations, the problem was solved with the high-fidelity model (using Navier-Stokes equations) alone using OPTIMA-2D. 85 design iterations are required to reduce the L_2 norm of the gradient by five orders of magnitude. In terms of CPU time, it is 5hrs 15min.

The variable fidelity method with both additive and multiplicative scaling methods is implemented on this problem. The multiplicative scaling method encountered some ill-conditioning which was countered by using an added constant in the multiplicative scaling method. The initial pressure distribution, the target pressure distribution (RAE 2822 airfoil) and the final

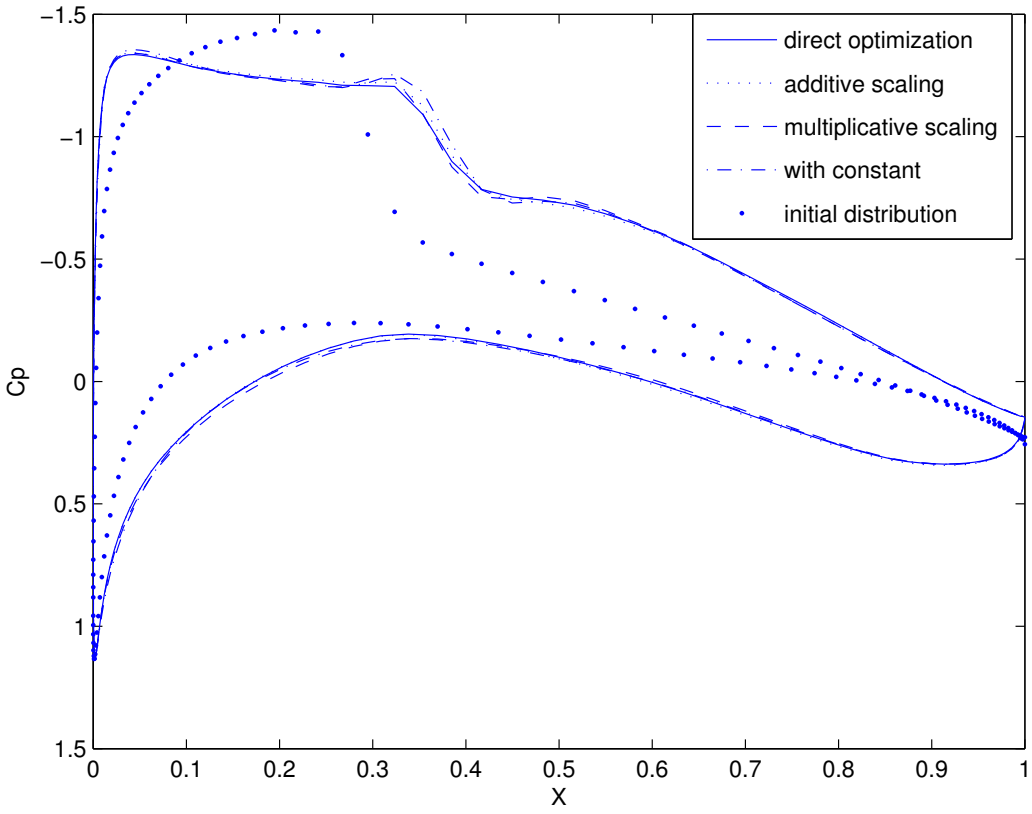

Figure 4.4: Pressure distribution for inverse design

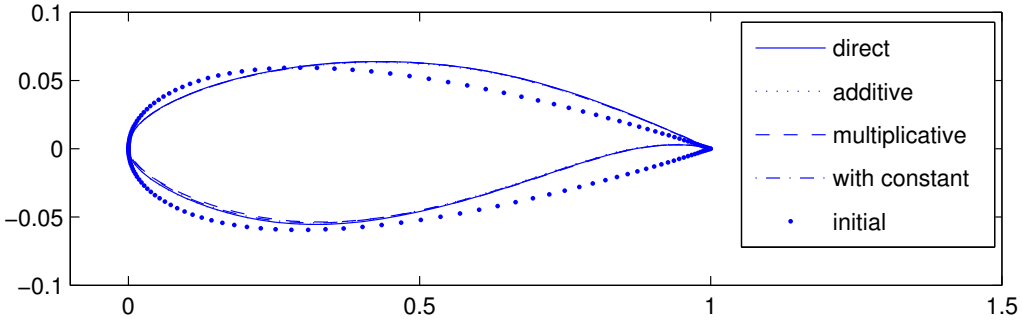

Figure 4.5: Airfoil shapes for inverse design

pressure distribution for all the scaling methods and their corresponding airfoil shapes are shown in Figure 4.5. Table 4.2 shows the number of high and low-fidelity function calls for both the variable fidelity frameworks and the function calls for the actual problem. The normalized area is the ratio of final area to the initial area between the C_p curves (actual and the target pressure curves). The C_p curves are difficult to distinguish because they all manage to match the target pressure distribution curve. However, the best objective value was obtained by direct optimization, 96.67% reduction compared to 96.53% reduction for additive scaling method and 94.21% for the multiplicative scaling method. The additive scaling method made only 5 calls to the high fidelity solver to converge while the direct optimization using high-fidelity made 85 calls. But the improvement in the computational time is not significant as the framework took 4hrs. This is due to two reasons, one reason being the time taken for a low-fidelity analysis is also significant and the other reason being solving the low-fidelity to unnecessarily high accuracy. Another thing to be considered is that as the framework converges towards the solution, the initial step size should be reduced. Otherwise, the scaled low-fidelity model will make many unnecessary iterations. The multiplicative scaling method made only 3 calls to the high-fidelity function and the framework achieved a two fold savings in computational time, but it managed only a 94% decrease in the objective function. This is because framework stopped without converging due to the ill-conditioning arising from the multiplicative scaling when the low fidelity objective function value is close to zero. So, in order to counter this problem, a constant K is added in the multiplicative scaling method. The multiplicative scaling method is tried for different values of K, starting from 0.001 and increasing to 0.10, and the best value is found for $K = 0.05$. The variable fidelity method using multiplicative scaling with an added constant managed to 95.3% decrease. Again, the framework stopped before it reached convergence, but the design point found was better than the multiplicative scaling method alone.

4.4.4 Maximization of Lift to Drag ratio

The subsonic design problem of maximizing the lift-to-drag ratio considered here is as presented in the Equation (4.7). The freestream conditions are are $M_\infty = 0.25$ and $Re = 2.88 \times 10^6$ and NACA 0012 is used as initial airfoil. The angle of attack is fixed during the optimization at

Table 4.2: Variable fidelity optimization results for inverse design problem

	Direct	Additive	Multiplicative	Constant
High Fidelity Function Calls	85	5	3	3
Low Fidelity Function Calls	–	241	147	153
Total CPU time	315 min	240 min	120 min	130 min
Normalized Area between the C_p curves	0.0325	0.0328	0.0342	0.0336

Table 4.3: Thickness constraints for the maximization of lift to drag ratio

Number	Position	Thickness
1	0.06	0.08
2	0.25	0.10
3	0.60	0.055
4	0.80	0.02
5	0.90	0.01

$\alpha = 9\,^\circ$. The airfoil shape is characterized by 15 B-spline control points and 10 control points are used as the design variables. As recommended by Nemec [11], five thickness constraints are specified. The high-fidelity and low-fidelity models are same as previous cases. The time taken for one optimization routine is 265 seconds for the high-fidelity case and around 45 seconds for the low-fidelity case. The high-fidelity model took 45 function evaluations to reduce the gradient by six orders of magnitude. In terms of computational time it took 130 min.

Again, the variable fidelity method was implemented on the problem with all the scaling methods. Table 4.4 shows the results for both additive and multiplicative type scaling methods. The initial pressure distribution and the final pressure distribution for all the scaling methods and their corresponding airfoil shapes are shown in Figures 4.6 and 4.7. The additive scaling method achieved the same results as the high-fidelity method. The additive scaling method took 90 min to converge while the direct optimization using high-fidelity took 130 min, which means a 30% improvement in the computational time. The framework for multiplicative scaling method stopped without converging because the scaled function could not find a better

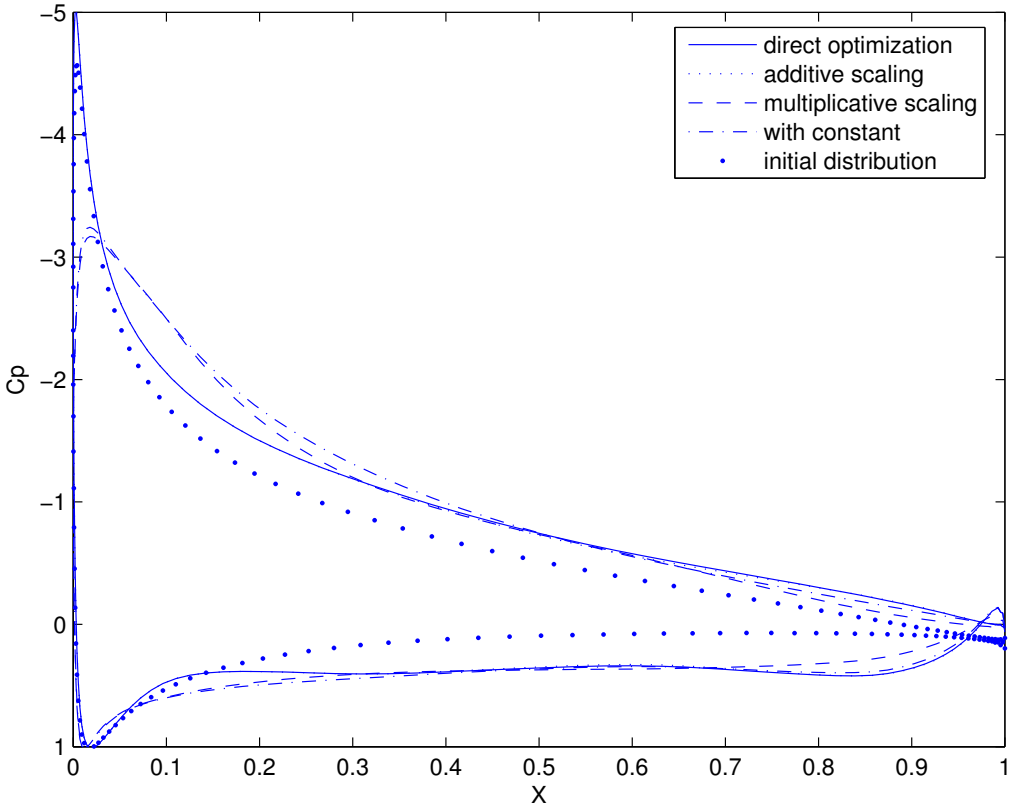

Figure 4.6: Pressure distribution for maximization of lift to drag ratio

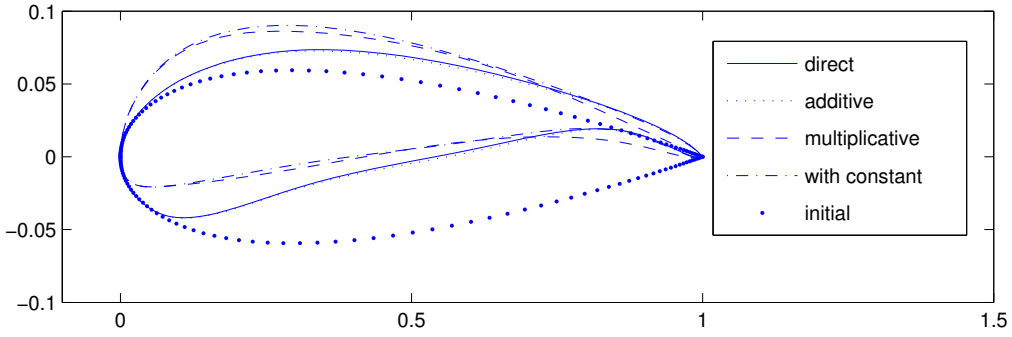

Figure 4.7: Airfoil shapes for maximum $C_L/\ C_D$

point. The multiplicative scaling method made 4 calls to the high-fidelity to converge while the direct optimization using high-fidelity made 45 calls. It achieved a 53% improvement in the computational time, but could not manage the same improvement in the objective function as shown in Table 4.4. So, a constant is added in the multiplicative scaling method to prevent this ill-conditioning. The multiplicative scaling method is again examined for different values of K, starting from 0.001 and sequentially increasing, and all the values achieved similar results though slightly better compared to the multiplicative scaling method. This is because the framework stopped as the the scaled function could not find a better point and not due to ill-conditioning. Also for these cases the optimization proceeded in a different direction, towards another local minimum, which is the reason for a different airfoil.

Table 4.4: Variable fidelity optimization results for max $C_L/\ C_D$ problem

	Direct Optimization	Additive	Multiplicative	with constant
C_L	1.3771	1.3770	1.4358	1.4469
C_D	0.0236	0.0235	0.0254	0.0255
High Fidelity Function Calls	45	6	3	4
Low Fidelity Function Calls	–	172	113	127
Total CPU time	130 min	90 min	60 min	70 min
$C_L/\ C_D$	58.22	58.21	55.74	56.47

4.5 Variable Fidelity Method with Different Physical Models

4.5.1 Overview

Although the variable fidelity method managed to save the computational time we felt it could be improved by using a lower fidelity model, as, the time taken for a function evaluation by a low-fidelity model is comparable to the time taken by the high-fidelity model. So, in order to use a lower fidelity model in the variable fidelity framework, we decided to use a different physical model as low-fidelity model.

Again, we considered aerodynamic optimization problems in the previous section as our

test cases. we used the same local scaling methods in the framework and it has been applied to inverse design optimization and lift-to-drag ratio maximization problems. The results are shown for these optimization problems in the following subsections.

4.5.2 Inverse Design

The problem here is the same as presented in the inverse design optimization problem in Section 4.6. The target pressure distribution is that of a RAE 2822 airfoil configuration and the initial pressure distribution is that of the NACA 0012 airfoil, as shown in figure 4.4. The freestream conditions are $M_\infty = 0.7$, $Re = 9 \times 10^6$ and angle of attack $\alpha = 3$. The airfoil shape is characterized by 15 B-spline control points and 12 control points are used as design variables (control points numbered 1, 8 and 15 are kept constant during the optimization). The high-fidelity model here, again, is a full Navier–Stokes solution with a turbulence model on a fine grid (283×65) while the low-fidelity model is an Euler solution over a coarse mesh (245×41). An optimization routine (a complete flow solution and gradient analysis) takes approximately couple of minutes for the fine grid while it takes about 15 seconds for the coarser grid.

The problem when solved with the high-fidelity model (using Navier-Stokes equations) alone using OPTIMA-2D took 85 design iterations to reduce the L_2 norm of the gradient by five orders of magnitude. In terms of CPU time, it is approximately 5hrs 15min.

The initial pressure distribution, the target pressure distribution (RAE 2822 airfoil) and the final pressure distribution for all the scaling methods and their corresponding airfoil shapes are shown in the Figure 4.9. Table 4.5 shows the results for both additive and multiplicative scaling methods. The table shows the number of high and low fidelity function calls for both the variable fidelity frameworks and the function calls for the actual problem. The C_p curves of direct method and additional scaling method are difficult to distinguish because they both manage to match the target pressure distribution curve. However, the best objective value was obtained by direct optimization, 96.67% reduction compared to 96.53% reduction for additive scaling method. But, the multiplicative scaling method could not achieve the same improvement. It could achieve only 94.21% and the framework stopped because it could not find a better point. The additive scaling method took 10 calls to the high fidelity to converge while the direct

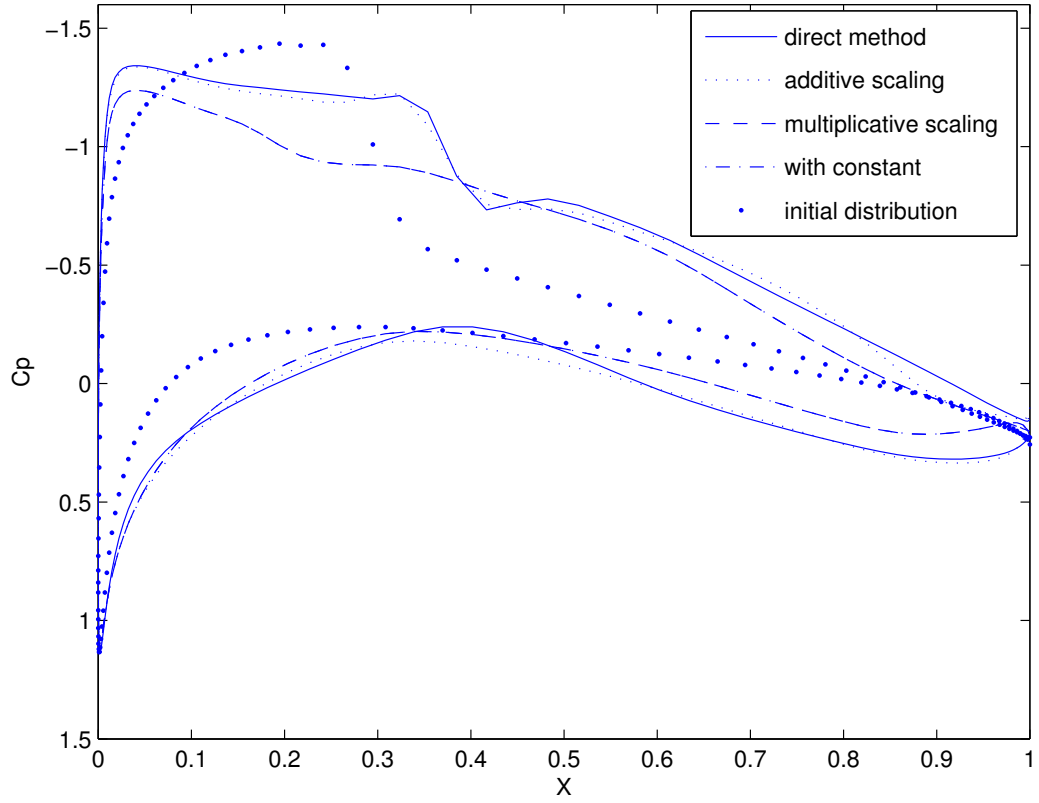

Figure 4.8: Pressure distribution for inverse design with different physical models

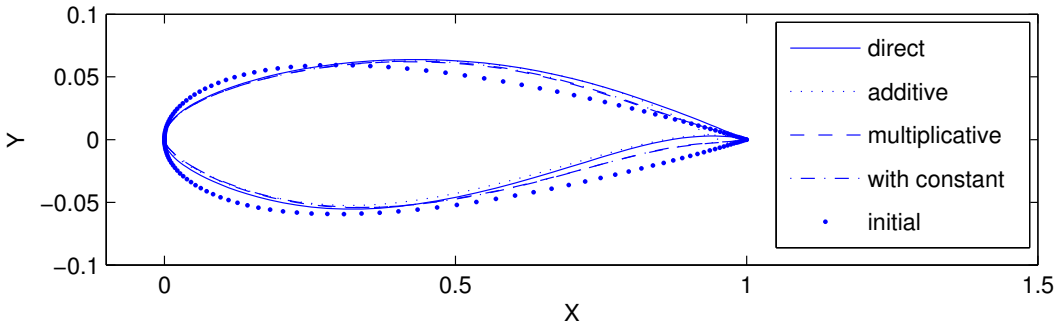

Figure 4.9: Airfoil shapes for inverse design with different physical models

Table 4.5: Variable physical fidelity optimization results for inverse design problem

	Direct	Additive	Multiplicative	constant
High Fidelity Function Calls	85	11	6	6
Low Fidelity Function Calls	–	279	137	143
Total CPU time	315 min	120 min	40 min	45 min
Normalized Area between the C_p curves	0.0325	0.0397	0.0694	0.0692

optimization using high-fidelity took 85 calls. This is because of the degree of matching between the low-fidelity and the high-fidelity model is not high as they are based on different physical models. And, the improvement in the computational time is significant as the framework took only around couple of hours. This is because the time taken for each function evaluation by the low-fidelity model is negligible compared to the hig-fidelity model. But the optimum achieved by the additive scaling method is a physically unrealistic airfoil as it violates the thickness constraints as can be seen in Figure 4.9. This can be attributed to the fact that we are using two totally different physical models. On the other hand, multiplicative scaling method achieved a physically realistic airfoil shape but its pressure distribution is not quite near the target airfoil's pressure distribution. It achieved only 94% decrease in the objective function. We also implemented the multiplicative scaling method with an added constant but the method achieved the same results as the multiplicative scaling method. Again, the framework stopped before it reached convergence and the design point found was the same as multiplicative scaling method.

4.5.3 Maximization of Lift to Drag ratio

The subsonic design problem of maximizing the lift to drag ratio considered here is, again, the same as presented in the section 4.6. The freestream conditions are are $M_\infty = 0.25$ and $Re = 2.88 \times 10^6$ and NACA 0012 configuration is used as initial airfoil. The angle of attack is fixed during the optimization at $\alpha = 9$. The airfoil shape is characterized by 15 B-spline control points and 10 control points are used as the design variables.

The high-fidelity and low-fidelity models are same as previous cases. The time taken for one

optimization routine is couple of minutes for the high-fidelity case and one-fourth of a minute for the low-fidelity case. But the variable fidelity framework did not manage to achieve the same kind of results as the previous cases. The framework did not converge for either of the scaling methods (additive or multiplicative kind). The reason being the flow solver did not converge for some of the intermediate airfoil shapes during the course of optimization routine. During the course of framework, the airfoil assumed extreme shapes for which the solver could not compute a complete flow solution. Due to this, we could not update the design variables and as a result we could not complete the variable fidelity framework algorithm. The reason for airfoil assuming extreme shapes can be attributed to the huge disparity between the euler and Navier-Stokes equations and as a result, between their respective objective functions and gradients. However, this problem was not encountered in the previous case (inverse design) as the objective function is the difference between the present pressure distribution and target pressure distribution, where both the high and low fidelity information are of the same order.

4.6 Variable Parameterization Method

For the variable parameterization case, the number of variables was varied during the optimization process, starting with low number of design variables, which means less accuracy of the shape parameterization but progressively increasing the number of design variables.

This method is applied to both inverse design and maximization of lift-to-drag ratio. A full Navier–Stokes solution with a turbulence model on a 201×45 grid is used and the freestream conditions are the same as in previous cases in section. For the inverse design problems, we started off with 9 design variables and performed optimization with more design variables while for lift-to-drag ratio maximization we started off with 8 design variables. The results can be seen in Table 4.6.

The method was tried for inverse design and maximization of lift-to-drag ratio. The number of iterations for problem with more design variables and which uses the optimum of the problem with fewer design variables as initial guess are shown in column n_{high} and the number of iterations for the actual problems with the same number of design variables are shown in

column n_{direct}. The time taken for the direct (actual) problem is shown in column $Time_{direct}$ while the time taken by the parameterization method is shown in $Time_{method}$. From the table we can see that, for some cases, using an approximate optimum as an initial guess results in fewer iterations, saving the computational time. For example, it achieved 13% decrease for the inverse design optimization problem with 20 design variables. However, it could not achieve the same reduction for the maximization of lift-to-drag ratio problem and the inverse design problem with 12 design variables. It actually increased the computational time by 20%. This is due to the fact that the computational time taken for each function evaluation is independent of the number of design variables. So, unless we can achieve savings in the total number of function evaluations (which is the case for the inverse design problem with 20 design variables) variable parameterization would be unable to produce savings in computational time.

Table 4.6: Variable parameterization computational cost comparison

	n_{direct}	$Time_{direct}$	n_{low}	n_{high}	$Time_{method}$
maximum $C_L/\ C_D$	52	34 min	17	44	43 min
Inverse Design (N=15)	83	43 min	26	68	51 min
Inverse Design (N=23)	153	76 min	26	133	68 min

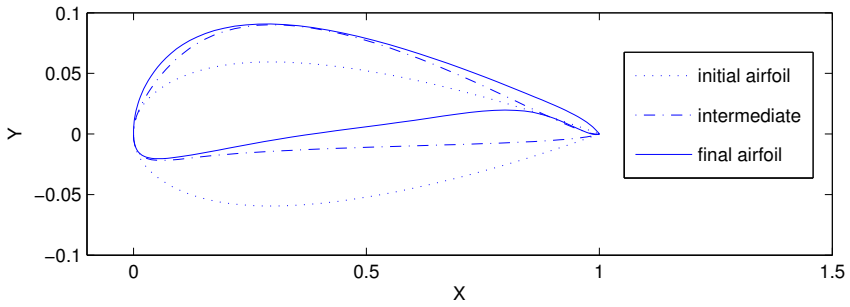

Figure 4.10: Variable parameterization for lift to drag ratio

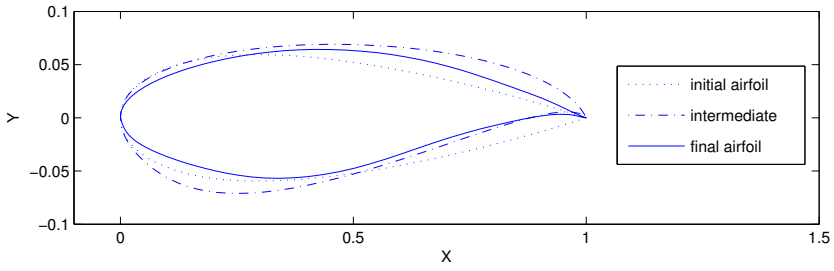

Figure 4.11: Variable parameterization for inverse design

Chapter 5

Conclusions and Recommendations

The studies made on the analytical case and the airfoil optimization problems gave us hope to reduce the computational effort required for an optimization process. The variable fidelity method performed well, which was expected, as it is a robust method. Both the scaling methods gave consistent results and succeeded in reducing the computational time. The variable parameterization method on the other hand is a crude method. A good insight of the problem is necessary to decide if the variable parameterization method will be efficient in reducing the computational time. Though further investigation is needed to determine the convergence properties of the method, variable parameterization method would perform well if used with good intuition as can be seen in the results of numerous sample problems that have been tried. A good mathematical study is needed to examine the convergence properties of the variable parameterization method.

Greater improvements can be achieved with the variable fidelity framework as well. For example, improvements can be made in the scaling methods to use higher order or global methods. And, we could change the convergence criteria for the sub-optimization process. The inner problem is solved to a higher degree of accuracy which is not needed as the algorithm only needs it ensure sufficient decrease in the high-fidelity problem. We could even adapt the quality of the inner problem as the algorithm proceeds, making it more accurate as it converges to the solution. Or, we could finish the final few steps with the high-fidelity method if the framework requires lots of high-fidelity evaluations as it approaches the optimum. However, in order to

improve and validate this framework, we need to conduct many more experiments. Then, this method can be implemented in the multidisciplinary framework.

Bibliography

[1] N. M. Alexandrov, J. E. Dennis, R. M. Lewis, and V. Torczon. A trust-region framework for managing approximations in optimization. *Journal of Structural Optimization*, 15(1):16–23, 1998.

[2] N. M. Alexandrov and R. M. Lewis. An overview of first-order model management for engineering optimization. *Journal of Optimization and Engineering,* 2, 413–430, 2001.

[3] N. M. Alexandrov, R. M. Lewis, L. L. Green, and P. Newman. Optimization with variable-fidelity models applied to wing design. AIAA Paper 2000-0841, Jan. 2000.

[4] N. M. Alexandrov, R. M. Lewis, C. R. Gumbert, L. L. Green, and P. Newman. Approximation and model management in aerodynamic optimization with variable-fidelity models. *Journal of Aircraft*, 38(6), Nov. 2001.

[5] N. M. Alexandrov, E. J. Nielsen, R. M. Lewis, and W. K. Anderson. First-order model management with variable-fidelity physics applied to multi-element airfoil optimization. AIAA Paper 2000-4886, Sept. 2000.

[6] J. Barthelemy and R. Haftka. Approximation concepts for optimum structural design - a review. *Journal of Structural Optimization*, 5:129–144, 1993.

[7] K. Chang, R. Haftka, G. Giles, and P. Kao. Sensitivity-based scaling for approximating structural response. *Journal of Aircraft*, 30(2):283–288, 1993.

[8] S. Gano, B. Sanders, and J. Renaud. Variable fidelity optimization using a kriging based scaling function. AIAA Paper 2004-4460, Sept. 2004.

[9] X. Marduel, C. Tribes, and J. Trepanier. Optimization using variable fidelity solvers: Exploration of an approximation management framework for aerodynamic shape optimization. AIAA Paper 2002-5595, Sept. 2002.

[10] J. R. R. A. Martins, P. Sturdza, and J. J. Alonso. The complex-step derivative approximation. *ACM Transactions on Mathematical Software*, 29(3):245–262, Sept. 2003.

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

[12] M. Nemec and D. W. Zingg. Newton-Krylov algorithm for aerodynamic design using the Navier-Stokes equations. *AIAA Journal*, 40(6):1146–1154, 2002.

[13] G. B. Thomas. *Calculus and Analytic Geometry (9th Edition)*. Addison Wesley Publishing Company, 1996.