On the Effect of Numerical Noise in Simulation-Based Optimization

Kay E. Vugrin

Thesis submitted to the Faculty of the
Virginia Polytechnic Institute and State University
in partial fulfillment of the requirements for the degree of

Master of Science
in
Mathematics

Jeff Borggaard, Chair
Gene Cliff
Terry Herdman
Shu-Ming Sun

March 12, 2003
Blacksburg, Virginia

Keywords: Shape Optimization, Sensitivity Analysis, Trust Region Algorithm

On the Effect of Numerical Noise in Simulation-Based Optimization

by
Kay E. Vugrin

ABSTRACT

Numerical noise is a prevalent concern in many practical optimization problems. Convergence of gradient based optimization algorithms in the presence of numerical noise is not always assured. One way to improve optimization algorithm performance in the presence of numerical noise is to adjust the method of gradient computation. This study investigates the use of Continuous Sensitivity Equation (CSE) gradient approximations in the context of numerical noise and optimization. Three problems are considered: a problem with a system of ODE constraints, a single parameter flow problem constrained by the Navier-Stokes equations, and a multiple parameter flow problem constrained by the Navier-Stokes equations. All three problems use adaptive methods in the simulation of the constraint and are numerically noisy. Gradients for each problem are computed with both CSE and finite difference methods. The gradients are analyzed and compared. The two flow problems are optimized with a trust region optimization algorithm using both sets of gradient calculations. Optimization results are also compared, and the CSE gradient approximation yields impressive results for these examples.

This material was based on work supported under a National Science Foundation Graduate Research Fellowship. Any opinions, findings, conclusions, or recommendations expressed in this publication are those of the author and do not necessarily reflect the views of the National Science Foundation.

Acknowledgments

I would like to thank my advisor, Jeff Borggaard, for his expertise and encouragement during the course of this project. I would also like to thank my committee members Gene Cliff, Terry Herdman, and Shu-Ming Sun for their insightful suggestions. The support of the National Science Foundation was greatly appreciated as I worked on this project. I am grateful to Dominique Pelletier for the use of his finite element software. Finally, I would like to thank my husband Eric for supporting me in all of my goals.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The field of numerical optimization is constantly seeking improved methods. In many cases, optimization algorithms produce sub-optimal solutions. Thus, an improved method that converges to an answer that is closer to optimal can result in tangible improvements. Many decisions in the world today rely upon the results of numerical optimization. Thus, even small improvements in standard optimization techniques can translate to substantial real world improvements. We are interested in an important class of optimization problems where the function value depends upon design parameters obtained through the solution of a physical model, often given as a differential equation. These problems are known as simulation-based optimization problems.

Gradient based optimization is a popular class of optimization methods. These algorithms are particularly advantageous in simulation-based optimization since the algorithms tend to find an optimum in relatively few function evaluations when compared to surrogate model or search based methods. The savings in function evaluations comes at a cost of requiring both function and gradient values at every iterate. Ideally, the function under consideration and its gradient are both known. However, there are many cases where the function is known only approximately, and the gradient is not known at all. Often, function values are computed with some degree of error. Function values may be too expensive to compute exactly, or the exact function value may be impossible to compute. For example, numerical simulations of high Reynolds number flows frequently cannot be accurately resolved without resorting to stabilization techniques [10]. Computational models of real world processes are not smooth [11]. Certain classes of problems can also result in the addition of numerical noise to function values. For instance, solutions to optimization problems including differential equations often involve the addition of noise due to adaptive methods, incomplete convergence, stabilization strategies, etc. [10] or non-smooth solutions [37]. When an inexactly described function is subject to optimization, certain problems can arise. Convergence is not always guaranteed. Also, if a function is not known, then its gradient is probably unknown. Thus, the gradient values used in optimization algorithms must be estimated or computed based on the inexact

function evaluations.

Inexact gradient information can have negative effects upon the convergence behavior of optimization algorithms [19]. For functions where only approximate discrete function evaluations are possible, there is generally a practical limit on the quality of the function evaluations. Thus, gradients that are computed based on function evaluations will also have inherent errors in value. The methods that are used to compute gradient values will affect the convergence properties of the optimization algorithm. Improved gradient estimation techniques could result in faster convergence or convergence to points closer to the true optimal value.

Finite difference methods are frequently used to compute necessary gradient information [32]. The gradient value is computed based upon the approximate function values. In other words, the function is approximated and then differentiated. The optimization algorithm uses the approximate function and its gradient as input. Thus, this approach results in a gradient that is "consistent" with the (approximated) function. Here, by consistent, we mean that the finite difference gradient approximates the derivative of the approximate function.

An alternative technique for gradient computation has recently come under investigation. This technique is based on differentiating the continuous form of the problem before any approximation occurs. This form of the gradient requires the derivative of the solution to the differential equation constraint with respect to the design parameter. This derivative can be found by using the implicit function theorem. The result is the continuous sensitivity equation (CSE). The required function and gradient values can then be computed by approximating the differential equation constraint and the CSE. This "differentiate-then-approximate" technique has roots in earlier works by Pironneau [39, 40] and Delfour and Zoleszio [20] and is demonstrated for shape derivatives for complex flows in [7], [9], [13], and [28]. When the CSE gradient is used, an approximation of the true gradient of the function is used in the optimization algorithm. The optimization algorithm uses approximate function values for the required function evaluations. Since the gradient of the approximate function and the approximate gradient of the original function are generally not the same, the gradient is said to be inconsistent with the function.

This study investigates the differences between the use of finite difference and CSE methods in the computation of gradient information. In particular, the two methods of gradient computation are compared in cases where function evaluations contain numerical noise. Three problems are considered. The first problem considered is an ODE constrained optimization problem. Numerical noise is introduced into function evaluations through the use of an adaptive time stepping algorithm. The gradient of the objective function is computed with both the finite difference and CSE methods, and the results are compared. The final two examples are taken from Burkardt, Gunzburger, and Peterson [13]. Viscous flow in a channel is approximated with an adaptive finite element method, where adaptation is based on both flow and sensitivity values. This problem has been investigated by Borggaard and Pelletier [9]. Two noisy objective functions are analyzed. Gradient information is computed with both the finite difference and CSE methods. Gradient information from both methods

is used by a trust region optimization algorithm, and convergence results are compared.

Chapter 2

Literature Review

Although advances in computational techniques have led to approximate solutions of many previously unsolvable problems, the investigation of optimization techniques under errors in objective function evaluation is not a new concern. Numerical noise from solution techniques is not the only cause of error in function evaluation. As early as the 1950s, statisticians were testing optimization algorithms on stochastic regression functions with uncertainties in function evaluation [34]. These early algorithms were not computationally intense, and they did not require the use of gradient based optimization techniques. By the late 1970s, more powerful optimization algorithms were under consideration. Glad and Goldstein proposed a forerunner to the modern trust region optimization algorithm and applied their algorithm to a function whose values could only be obtained with an error [26]. They also computed error limits on gradient and Hessian values based on the bounded function evaluation error and proved that the algorithm converged to a certain accuracy based on the magnitude of the error in the function evaluations.

Study of optimization under errors in the objective function has since exploded. In particular, the advent of numerical approximation techniques for differential equations has led to many interesting optimization problems with noisy objective functions. Problems with approximate solutions to differential equations in the objective function arise frequently, as do differentially constrained optimization problems. Many practical optimization problems require the minimization of an objective function $\mathcal{J}(\cdot)$ that depends upon design parameters obtained through solutions of differential equations. The objective functions are subject to numerical noise. Thus, several issues arise. Will an optimization algorithm converge in the event of errors in the objective function? What are the convergence properties? If a gradient based method is used, how can the gradient be computed? How much computational time and effort is required for convergence?

Non-gradient based optimization is an area that attracts research interest. For recent investigations, see [1, 47, 18, 11, 48]. In fact, non-gradient based algorithms may be less susceptible to the effects of numerical noise. When gradient based algorithms work, however, they often

return results with less computational effort. Thus, the interest of this study is gradient based optimization in the presence of numerical noise.

2.1 Sensitivity Analysis

A common issue that arises in the use of gradient based optimization algorithms is the question of how to compute gradient information. Closed-form solutions of differential equation constraints are often unknown. Thus, approximations are used to evaluate function values. The true objective function $\mathcal{J}(\cdot)$ and the approximate function $\mathcal{J}^N(\cdot)$ are two different functions. In many cases, sensitivity analysis refers to the computation of the gradient of either $\mathcal{J}(\cdot)$ or $\mathcal{J}^N(\cdot)$. In our case, we refer to sensitivity analysis as the technique of computing the gradient by implicitly differentiating the constraints (continuous or discrete).

There are two general approaches to sensitivity analysis. The difference between the two approaches is the order of the operations: approximation and differentiation. In the discrete approach, the constraints are approximated first, then differentiation is applied to the discrete equations. The approximation algorithm can be differentiated by one of several methods. For instance, automatic differentiation, hand differentiation of the code, and the application of finite differences are all ways to compute $\nabla\mathcal{J}^N(\cdot)$ [29]. In the continuous approach, however, the implicit function theorem is applied to the original constraint equation. The coupled equations are then solved simultaneously, resulting in an approximation of the objective function $\mathcal{J}(\cdot)$ and an approximation of of its gradient $(\nabla\mathcal{J}(\cdot))^N$. The continuous approach results in inconsistent gradients: $(\nabla\mathcal{J}(\cdot))^N$ is not generally equal to $\nabla\mathcal{J}^N(\cdot)$. Thus, the optimization algorithm will attempt to minimize $\mathcal{J}^N(\cdot)$ using a gradient that is in fact an approximation to $\nabla\mathcal{J}(\cdot)$. Hou et al. demonstrate optimization failure when inconsistent derivatives are used in [30].

The discrete approach approximates the gradient of the approximate function $\mathcal{J}^N(\cdot)$. Thus, the optimization algorithm will attempt to minimize $\mathcal{J}^N(\cdot)$ while using gradient values that are consistent with that function. Although the most popular discrete approach is the use of finite difference schemes, there are several drawbacks to this strategy [5]. Finite difference schemes calculate the gradient information through function evaluations, and this can be expensive. Additionally, gradient information computed with finite difference schemes can be conflicting or inaccurate [10]. The discrete approach computes the derivative of noise in $\mathcal{J}^N(\cdot)$, and convergence problems can result [25]. Thus, many researchers have investigated alternate methods of both continuous and discrete gradient computation. Gill et al. investigate errors when different finite difference schemes are used to compute gradients [24]. Taylor et al. investigate the difference between finite difference schemes and direct differentiation schemes for computation of gradient information that is supplied to an Automated Design Synthesis Optimization program [45]. Burkardt et al. propose a discrete sensitivity system and analyze the effectiveness of this method [12, 13]. Taylor et al. also investigate direct differentiation and adjoint variable methods [44, 31, 45]. Stanley compares the hybrid

sensitivity equation method and abstract semi-analytical method in the context of a trust region algorithm [43]. In the hybrid sensitivity equation method, a sensitivity equation is derived through implicit differentiation of the constraints. The sensitivity equation is transformed and solved, and the solution is mapped back to the physical domain. In the abstract semi-analytical method, the infinite dimensional transformed state equation is differentiated in order to find an equation for the sensitivity of the transformed state.

2.2 Shape Optimization and Shape Sensitivities

In some types of optimal design problems, the method of shape optimization is quite useful. Shape optimization is used in aircraft, spacecraft, and structural design, among other applications. Essentially, shape optimization problems involve optimizing an objective function that is based on physical parameters that describe a physical shape, subject to constraint criteria [36]. Often, the objective function takes the form of an integral over the physical domain or its boundary. The integrand usually depends smoothly upon the solution of a boundary value problem [42].

Shape optimization problems give rise to shape sensitivities. Shape parameters are values that affect the location, orientation, or boundary characteristics of the problem domain. The computation of shape sensitivities, i.e. how the state variables of a system change relative to changes in shape parameters, is not always straightforward. In the discrete approach, the total derivative of the state variable with respect to the shape parameter is approximated. In the continuous approach, a partial derivative of the state variable with respect to the shape parameter is approximated [50]. During the implementation of the discrete approach, mesh sensitivities are introduced. Mesh sensitivities describe how mesh coordinates tend to move as the parameters are varied. They are often computed by using the difference between two meshes (with the same topology) but can be difficult to quantify in may cases. An advantage of the continuous approach is that mesh sensitivities are avoided entirely.

Shape optimization, shape sensitivities and associated mesh sensitivities are actively studied and debated. Hou et al. have examined the effects of internal nodal movements on shape design sensitivity [30]. Ito et al. investigate optimal shape design for a crystal growing reactor and compute material and shape derivatives to the Boussinesq approximation in [33]. Stanley investigates shape sensitivity computation in a heat equation problem [43]. Borggaard et al. study shape optimization and shape sensitivities in the context of flow problems [3, 8]. Taylor et al. discuss the issue of shape sensitivities in a series of papers [44, 45, 31]. A short list of other research efforts in this area are given in references [12, 13, 31].

2.3 Continuous Sensitivity Equation Method

There has been a great deal of interest in the so called Continuous Sensitivity Equation (CSE) method. Borggaard et al. began investigating the use of the the CSE method in [7]. The technique computes an equation for sensitivities by differentiating the infinite dimensional constraint equation and then discretizing the result. The CSE method avoids the issue of mesh sensitivities when shape sensitivities are computed. Borggaard and Burns have also investigated convergence of trust region algorithms supplied with asymptotically consistent gradients computed with the CSE method [5]. Trust region algorithm results are compared when CSE and other gradient computation techniques are used in [4, 9, 49, 8]. This study continues analysis of the CSE method combined with optimization algorithms.

2.4 Trust Region Algorithms

There are many different gradient based optimization algorithms that have been analyzed in the presence of numerical noise in the objective function. Augmented Lagrangian Methods [33], Automated Design Synthesis [45], and the Method of Feasible Directions [14] are only a few of the many gradient based techniques that have been studied for problems with numerical noise. Trust region optimization algorithms, however, are particularly interesting when applied to problems with numerical noise. Trust region algorithms are able to perform well, even in cases of inaccurate function and gradient information [15, 16, 17, 46]. Convergence of trust region algorithms that use inaccurate function and gradient information has been proven under certain circumstances [22, 16, 6]. Numerical investigations of this topic are frequent; [43, 17, 5, 9, 6, 4, 10] is a short list of papers analyzing trust region algorithm performance in the presence of noise.

Newton's method of optimization is quadratically convergent only within small neighborhoods of the stationary point [2]. Thus, quasi-Newton methods have evolved with a goal of global convergence. The trust region algorithm is one such method. Trust region algorithms can vary widely, but they share common characteristics. Trust region algorithms are iterative processes that use previously computed function values to estimate a region over which the current quadratic model can be used to accurately represent the function. For each step k in the iteration, a quadratic model function m_k that attempts to approximate the objective function is computed using approximate Hessians. Each iteration has a corresponding trust region radius δ_k, within which the model function m_k is trusted as an approximation of the actual objective function $\mathcal{J}^N$. Thus, the next iterate a_{k+1} is computed by minimizing the model within the trust region:

$$m_k(a_{k+1}) = \min_{\|s_k\| \leq \delta_k} m_k(a_k + s_k) = \min_{\|s_k\| \leq \delta_k} (\mathcal{J}^N(a_k) + \nabla \mathcal{J}^N(a_k)^T s_k + \frac{1}{2} s_k^T H_k s_k). \qquad (2.1)$$

The algorithm computes a step size no larger than the trust region radius. If the objective function value has improved sufficiently, this step is accepted, and perhaps the trust region radius is increased for the next iteration. If the objective function does not improve sufficiently, the step is rejected and the trust region radius will be reduced [19]. The following algorithm outlines a trust region algorithm [21].

Select an initial guess $a_0 \in \mathcal{A}$, an initial trust-region radius δ_0 and constants $0 < \eta_1 < \eta_2 < 1$ and $0 < \gamma_1 < 1 < \gamma_2$. Compute $\mathcal{J}^N(a_0)$, $\nabla\mathcal{J}^N(a_0)$ and select or initialize H_0.

Do $k = 0, 1, \ldots$, until "convergence":

1. Determine the approximate solution s_k to equation (2.1). For a small number of design parameters, we solve problem 2.1 using a line search.
2. Let $\rho_k = \frac{\mathcal{J}^N(a_k) - \mathcal{J}^N(a_{k+1})}{m_k(a_k) - m_k(a_{k+1})}$.
3. If $\rho_k < \eta_1$, then set $\delta_{k+1} \in (0, \gamma_1\delta_k]$ and $a_{k+1} = a_k$, $\mathcal{J}^N(a_{k+1}) = \mathcal{J}^N(a_k)$, $\nabla\mathcal{J}^N(a_{k+1}) = \nabla\mathcal{J}^N(a_k)$ and $H_{k+1} = H_k$.
4. If $\eta_1 < \rho_k < \eta_2$, then set $\delta_{k+1} \in (0, \delta_k]$ and $a_{k+1} = a_k + s_k$. Compute $\mathcal{J}^N(a_{k+1})$, $\nabla\mathcal{J}^N(a_{k+1})$ and the update H_{k+1}.
5. If $\eta_2 < \rho_k$, then set $\delta_{k+1} \in [\delta_k, \gamma_2\delta_k]$ and $a_{k+1} = a_k + s_k$. Compute $\mathcal{J}^N(a_{k+1})$, $\nabla\mathcal{J}^N(a_{k+1})$ and the update H_{k+1}.

Continue

For our optimization results, we choose the following values for these parameters: $\eta_1 = 0.1$, $\eta_2 = 0.9$, $\gamma_1 = 0.5$, and $\gamma_2 = 2.0$. The small value for η_1 reflects the fact that we place a high premium on avoiding extra function evaluations. Hessian updates are performed by the BFGS secant update [21].

Chapter 3

Model Equations

3.1 ODE Example

The first example that we consider is a differentially constrained optimization problem originally proposed by Gockenbach and Symes [27]. The following system of ODEs describes the orbit of a satellite of the earth-moon system. This system of equations is also example `orbitode` in MATLAB demonstrating use of the built-in function `ode23`. Gockenbach and Symes examine this problem in order to investigate numerical noise in the solutions of differential equations. This example seeks to find the optimal relative mass of the moon, a, so that the orbit is periodic with period T. The problem is formulated as follows:

Find the positive parameter a that minimizes the function

$$\mathcal{J}(a) = \frac{1}{2}\left[(x(T;a) - x_0)^2 + (x'(T;a) - x_0')^2 + (y(T;a) - y_0)^2 + (y'(T;a) - y_0')^2\right] \tag{3.1}$$

subject to $(x(\cdot;a), y(\cdot;a))$ solving the second order system

$$x''(t;a) = 2y'(t;a) + x(t;a) - \frac{(1-a)(x(t;a)+a)}{r_e(x,y)^3} - \frac{a(x(t;a)-1+a)}{r_m(x,y)^3} \tag{3.2}$$

$$y''(t;a) = -2x'(t;a) + y(t;a) - \frac{(1-a)y(t;a)}{r_e(x,y)^3} - \frac{ay(t;a)}{r_m(x,y)^3} \tag{3.3}$$

with

$$x_0 = x(T;0) = 1.2, \qquad y_0 = y(T;0) = 0, \tag{3.4}$$

$$x_0' = x'(T;0) = 0, \qquad y_0' = y'(T;0) = -1.04935751. \tag{3.5}$$

The final time $T = 6.19216933$, $r_e(x,y)$ is the distance from (x,y) to $(-a,0)$, and $r_m(x,y)$ is the distance from (x,y) to $(1-a,0)$.

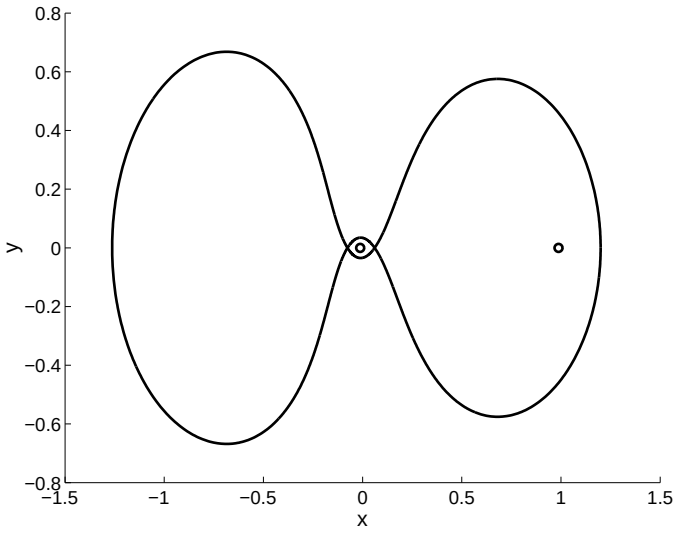

Figure 3.1: The Orbit of the Satellite Modelled by the ODE System

3.2 Flow Example: Single Parameter

Our flow examples are based on a viscous channel flow example used by Burkardt, Gunzburger, and Peterson [13]. The channel under consideration is a two-dimensional channel with a bump along the bottom. The channel has unit height and is 10 units long. In the single parameter flow example, a single parameter a determines the shape of the obstruction along the bottom of the channel. The parameter a is used to describe the shape of the bottom wall through the following equation:

$$y(x,a) = \frac{a}{2}\left[1 - \cos\left((x-2)\pi\right)\right] \quad x \in (2,4). \tag{3.6}$$

The obstruction blends smoothly into the channel at both ends. The problem that we consider on this channel is as follows:

Find the shape parameter $a \in (-0.2, 0.25)$ that minimizes the function

$$\mathcal{J}(a) = \int_0^1 \left(4y(1-y) - u(5,y;a)\right)^2 \, \mathrm{d}y, \tag{3.7}$$

subject to the horizontal velocity u satisfying the Navier-Stokes equations

$$\mathbf{u}\cdot\nabla\mathbf{u} = -\nabla p + \frac{1}{Re}\triangle\mathbf{u} \tag{3.8}$$

$$\nabla\cdot\mathbf{u} = 0 \tag{3.9}$$

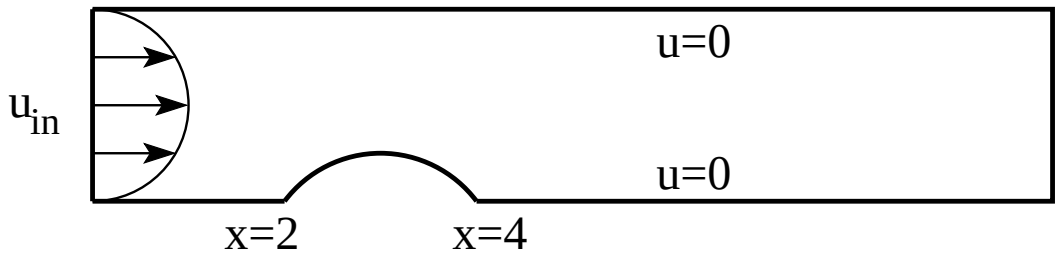

Figure 3.2: Single Parameter Flow Example: Initial and Boundary Flow Conditions

where the non-dimensional flow quantities are the velocity vector $\mathbf{u} = (u, v)$ and the pressure p. As in [13], the Reynolds number, Re, is set to one for this study. For boundary conditions, we impose the fully developed channel flow:

$$u(0, y; a) = 4y(1 - y) \tag{3.10}$$

at the inflow, zero velocity at the walls, and zero stress at the outflow. The objective function is the difference between straight channel flow and the obstructed flow one unit width downstream of the obstruction. This problem is constructed to clearly identify the minimum at $a = 0$, where the channel wall is flat. Figures 3.3 and 3.4 are illustrations of channel geometry for two a values.

Figure 3.3: Single Parameter Flow Example: Channel Domain, $a = 0.25$

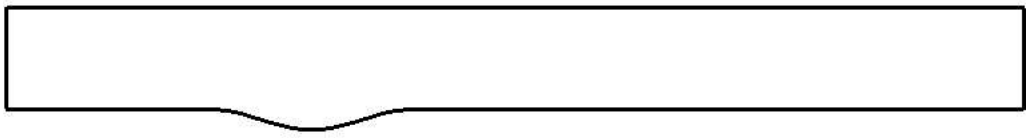

Figure 3.4: Single Parameter Flow Example: Channel Domain, $a = -0.20$

3.3 Flow Example: Multiple Parameters

The channel flow problem is again considered. However, the shape of the obstruction in the channel is controlled by three parameters: a_1, a_2, and a_3. The equation for the shape of the obstruction is:

$$y(x, a_1, a_2, a_3) = \sum_{i=0}^{4} B_{i,4}(x) a_i \quad x \in (2, 4) \tag{3.11}$$

where $a_0 = 0$, $a_4 = 0$, and

$$B_{i,4}(x) = \frac{4!}{i!(4-i)!}(x-2)^i(4-x)^{4-i}, \tag{3.12}$$

where $B_{i,4}(x)$ are the Bezier curves such that $B_{i,4}(2) = B_{i,4}(4) = 0$, $i = 0, \ldots, 4$ [38]. The height of the obstruction is very sensitive to the magnitude of $\vec{a}$. Thus, we restrict the values of $\vec{a}$ to a smaller range than in the single parameter flow example. The design problem becomes:

Find the shape parameters $a_1 \in (-.02, .02)$, $a_2 \in (-.02, .02)$, $a_3 \in (-.02, .02)$ that minimize the function

$$\mathcal{J}(a_1, a_2, a_3) = \int_0^1 \left(4y(1-y) - u(5, y; a_1, a_2, a_3)\right)^2 \, \mathrm{d}y. \tag{3.13}$$

The same initial conditions, boundary conditions, and constraints are evaluated on this channel as in the previous example. The objective function is still the difference between straight channel flow and the obstructed flow one unit width downstream of the obstruction. This problem is constructed to clearly identify the minimum at $a_1 = 0$, $a_2 = 0$, $a_3 = 0$. Figures 3.5 through 3.7 are illustrations of channel geometry for three $\vec{a}$ values.

Figure 3.5: Multiple Parameter Flow Example: Channel Domain, $\vec{a} = (0.02, 0.02, 0.02)$

Figure 3.6: Multiple Parameter Flow Example: Channel Domain, $\vec{a} = (-0.02, 0.02, 0.02)$

Figure 3.7: Multiple Parameter Flow Example: Channel Domain, $\vec{a} = (0.02, 0.02, -0.02)$

Chapter 4

Methods

4.1 ODE Example

4.1.1 Objective Function Evaluation

An adaptive solution method allows the time step to vary as the solution evolves. This property is useful in this example, since the direction of the satellite changes very quickly as the satellite nears earth. A small time step is required for accurate integration during this part of the orbit. When the satellite is not near the earth, a larger time step still allows for the integration to be performed efficiently [27].

The adaptive time stepping solver `ode23` is thus a good choice to solve the ODE constraint system. The ODEs are solved in MATLAB using the built-in function `ode23` with three error tolerances as specified in Table 4.1. MATLAB's default tolerances are `rel_tol` $= 1 \times 10^{-3}$ and `abs_tol` $= 1 \times 10^{-6}$. Thus, cases two and three have tighter tolerances than the default. $\mathcal{J}^N(a)$ is computed for each of the three error tolerance cases based on the values of $x^N(T;a)$, $y^N(T;a)$, $(x'(T;a))^N$, and $(y'(T;a))^N$ returned by `ode23`:

$$\begin{aligned}\mathcal{J}^N(a) &= \frac{1}{2}[(x^N(T;a)-x_0)^2+((x'(T;a))^N-x_0')^2+ \\ &\quad (y^N(T;a)-y_0)^2+((y'(T;a))^N-y_0')^2]. \end{aligned} \tag{4.1}$$

In order to generate plots of $\mathcal{J}^N(a)$ for each of the three cases, the objective function is evaluated for $a \in [0.012, 0.01255]$ for $\Delta a = 10^{-6}$.

Table 4.1: List of `ode23` Tolerances

	`rel_tol`	`abs_tol`
Case 1	1×10^{-3}	1×10^{-5}
Case 2	1×10^{-4}	1×10^{-6}
Case 3	1×10^{-6}	1×10^{-9}

4.1.2 Finite Difference Gradient

The finite difference gradients are computed with the forward-difference scheme:

$$\frac{d\mathcal{J}^N(a)}{da} \approx \frac{\mathcal{J}^N(a+\Delta a) - \mathcal{J}^N(a)}{\Delta a}, \tag{4.2}$$

where $\Delta a = 10^{-6}$ [41]. The derivative, $\frac{d\mathcal{J}^N(a)}{da}$, is computed for the same a range and Δa value as above for graph generation.

4.1.3 CSE Gradient

CSE gradient approximations are also computed. The objective function is differentiated with respect to a:

$$\begin{aligned}\frac{d\mathcal{J}(a)}{da} &= (x(T;a) - x_0)\frac{\partial x(T;a)}{\partial a} + (x'(T;a) - x'_0)\frac{\partial x'(T;a)}{\partial a} \\ &\quad + (y(T;a) - y_0)\frac{\partial y(T;a)}{\partial a} + (y'(T;a) - y'_0)\frac{\partial y'(T;a)}{\partial a}.\end{aligned} \tag{4.3}$$

The four sensitivity quantities are computed by implicit differentiation of the ODE constraints (Equations 3.2 and 3.3) along with the initial conditions. For the theory behind implicit differentiation, see [35]. When implicit differentiation is complete, the following system of equations results:

Let $s_x = \frac{\partial x(T;a)}{\partial a}$, $s_y = \frac{\partial y(T;a)}{\partial a}$, then

$$\begin{aligned}\frac{\partial^2}{\partial t^2}s_x &= 2\frac{\partial s_y}{\partial t} + s_x - \frac{r_e[(1-a)(s_x+1) - (x+a)] - 3r'_e(1-a)(x+a)}{r_e^4} \\ &\quad - \frac{r_m[x-1+a+a(s_x+1)] - 3r'_m a(x-1+a)}{r_m^4},\end{aligned} \tag{4.4}$$

$$\frac{\partial^2}{\partial t^2}s_y = -2\frac{\partial s_x}{\partial t} + s_y - \frac{r_e[(1-a)s_y - y] - 3r'_e(1-a)y}{r_e^4} - \frac{r_m[y + as_y] - 3r'_m ay}{r_m^4}, \tag{4.5}$$

where the initial conditions are zero:

$$s_x(0) = 0, \qquad \frac{\partial s_x(0)}{\partial t} = 0, \qquad s_y(0) = 0, \quad \text{and} \qquad \frac{\partial s_y(0)}{\partial t} = 0. \tag{4.6}$$

(For brevity's sake, the arguments are not included in the above system.)

Since the system 4.4–4.5 contains arguments of $x(t)$ and $y(t)$, we use MATLAB's `ode23` to solve this system of equations and the system of constraints simultaneously. The resulting values are used to compute $(\frac{d\mathcal{J}(a)}{da})^N$. The CSE gradient approximation is computed and graphed on the same interval as the objective function.

4.2 Flow Example: Single Parameter

4.2.1 Flow Solver

An adaptive finite element strategy is used to approximate the flow and continuous sensitivity equations. The approximate equations are solved using Crouzeix- Raviart triangular elements. These elements allow for piecewise quadratic approximations for the velocity and discontinuous piecewise bilinear approximations for the pressure. The incompressibility constraint is treated with an augmented Lagrangian technique. The result of the approximation is a set of nonlinear algebraic equations, which are solved using Newton's method. For a given mesh, the resulting nonlinear system is solved for the quantities u^N and $(\frac{\partial u}{\partial a})^N$. The initial mesh is very coarse, and the initial guess for the solution is zero. An adaptive mesh refinement strategy is combined with the finite element algorithm. An estimate of the error on the current mesh is used to help design a new mesh that reduces the error estimate by a factor of two. Computations on the new mesh result in a better estimate for the error. Thus, solving the same flow problem over iterated meshes results in successively better meshes where solutions are interpolated from the previous mesh. When CSE gradients are computed, adaptivity takes into account error estimates for the flow as well as sensitivity variables. When the flow solver is used to compute the finite difference gradients, adaptivity only considers error estimates for the flow variables [9].

4.2.2 Objective Function Evaluation

The objective function $\mathcal{J}(a)$ is approximated based on the results of the flow solver. Recall that

$$\mathcal{J}(a) = \int_0^1 \left(4y(1-y) - u(5, y; a)\right)^2 \, \mathrm{d}y. \tag{4.7}$$

As in Burkardt et al. [13], the integral above is approximated by the composite trapezoid rule. In general:

$$\int_a^b f(x)\mathrm{d}x \approx h * (\frac{1}{2}f_0 + f_1 + \ldots + f_{n-1} + \frac{1}{2}f_n) \tag{4.8}$$

where $h = \frac{b-a}{n}$ [23].

We take $n = 10$ and $h = 0.1$. Recall that $u^N(x,0) = u(x,0) = 0$ and $u^N(x,1) = u(x,1) = 0$ from our prescribed boundary conditions. Thus, our objective function approximation is

$$\mathcal{J}^N(a) = h[(4y_1(1-y_1) - u^N(5,y_1;a))^2 + \ldots + (4y_9(1-y_9) - u^N(5,y_9;a))^2] \tag{4.9}$$

where $y_1 = 0.1$, $y_2 = 0.2$, $\ldots y_8 = 0.8$, $y_9 = 0.9$.

Thus, for each objective function evaluation, we need to determine the value of u^N at nine coordinate points. Since the flow solver returns u^N values over triangular elements, we transform the $(5, y)$ coordinate pair into barycentric coordinates and use quadratic interpolation over the element that contains each point. This computation returns $u^N(5, y; a)$, and $\mathcal{J}^N(a)$ is then evaluated.

In order to graph $\mathcal{J}^N(a)$ for each adaptation, $\mathcal{J}^N(a)$ is computed for $a \in [-0.2, 0.25]$, $\Delta a = 0.0001$ for each of four mesh adaptations.

4.2.3 Finite Difference Gradients

Two finite difference methods are used to compute gradients for this example [41]. The forward difference method is used:

$$\frac{d\mathcal{J}^N(a)}{da} \approx \frac{\mathcal{J}^N(a+\Delta a) - \mathcal{J}^N(a)}{\Delta a} \tag{4.10}$$

and the central difference method is used:

$$\frac{d\mathcal{J}^N(a)}{da} \approx \frac{\mathcal{J}^N(a+\Delta a) - \mathcal{J}^N(a-\Delta a)}{2\Delta a} \tag{4.11}$$

with Δa values of 0.01, 0.001, 0.0001, and 0.00001.

4.2.4 CSE Gradient

The CSE gradient $(\frac{d\mathcal{J}(a)}{da})^N$ is computed in a manner similar to $\mathcal{J}^N(a)$. The objective function is first differentiated with respect to a:

$$\frac{d\mathcal{J}(a)}{da} = \int_0^1 -2(4y(1-y) - u(5,y;a))\frac{\partial u(5,y;a)}{\partial a}dy. \tag{4.12}$$

The flow solver returns u^N and $(\frac{\partial u}{\partial a})^N$ on triangular elements, so conversion to barycentric coordinates is again necessary. Once u^N and $(\frac{\partial u}{\partial a})^N$ have been computed at the nine necessary points, the composite trapezoid rule is used to evaluate the integral. Thus,

$$\left(\frac{d\mathcal{J}(a)}{da}\right)^N = h\left[-2(4y_1(1-y_1) - u^N(5,y_1;a))\left(\frac{\partial u(5,y_1;a)}{\partial a}\right)^N + \ldots \right.$$
$$\left. - 2(4y_9(1-y_9) - u^N(5,y_9;a))\left(\frac{\partial u(5,y_9;a)}{\partial a}\right)^N\right], \qquad (4.13)$$

where $y_1 = 0.1$, $y_2 = 0.2$, $\ldots y_8 = 0.8$, $y_9 = 0.9$.

In order to graph $(\frac{d\mathcal{J}(a)}{da})^N$ for each adaptation, the CSE gradient approximation is computed for $a \in [-0.2, 0.25]$, $\Delta a = 0.0001$ for each of the four adaptations.

4.2.5 Trust Region Algorithm

A trust region algorithm is applied to the flow problem using each of the two gradient computation methods. A secant optimization method is used. Convergence criteria is either a gradient value or step value less than 10^{-11}. The initial trust region radius is 0.1, and a value of $\Delta a = 0.01$ is used for computation of the forward difference gradient approximation. The central difference gradient approximation is not considered. Since we are using a very low Reynolds number ($Re = 1$), the flow quickly develops a quadratic profile. Thus, the objective function is very flat, and some scaling is needed. The function and gradient values are scaled by a factor of 10^5. The flow solution after three mesh adaptations is used during optimization.

4.3 Flow Example: Multiple Parameters

4.3.1 Flow Solver

The same flow solver is used for both the single and multiple parameter flow problems. In the multiple parameter case, the flow solver returns values for u^N, $(\frac{\partial u}{\partial a_1})^N$, $(\frac{\partial u}{\partial a_2})^N$, and $(\frac{\partial u}{\partial a_3})^N$.

4.3.2 Objective Function Evaluation

The objective function is evaluated in the same way as the single parameter example. The flow solver returns $u^N(x, y; a_1, a_2, a_3)$. Through the same process outlined earlier, the (x, y)

coordinates are transformed into barycentric coordinates and u^N is computed at each necessary point. Thus,

$$\begin{aligned}\mathcal{J}^N(a_1,a_2,a_3) &= h[(4y_1(1-y_1)-u^N(5,y_1;a_1,a_2,a_3))^2+\ldots \\ &\quad +(4y_9(1-y_9)-u^N(5,y_9;a_1,a_2,a_3))^2],\end{aligned} \tag{4.14}$$

where $y_1 = 0.1$, $y_2 = 0.2$, $\ldots y_8 = 0.8$, $y_9 = 0.9$.

4.3.3 Finite Difference Gradient

The forward difference method is used to compute the gradient for this example:

$$\frac{\partial}{\partial a_1}\mathcal{J}^N(a_1,a_2,a_3) = \frac{\mathcal{J}^N(a_1+\Delta a,a_2,a_3)-\mathcal{J}^N(a_1,a_2,a_3)}{\Delta a}. \tag{4.15}$$

The same approximation is used for $\frac{\partial}{\partial a_2}\mathcal{J}^N(a_1,a_2,a_3)$ and $\frac{\partial}{\partial a_3}\mathcal{J}^N(a_1,a_2,a_3)$.

4.3.4 CSE Gradient

The CSE gradient components $(\frac{\partial}{\partial a_1}\mathcal{J}(a_1,a_2,a_3))^N$, $(\frac{\partial}{\partial a_2}\mathcal{J}(a_1,a_2,a_3))^N$, and $(\frac{\partial}{\partial a_3}\mathcal{J}(a_1,a_2,a_3))^N$ are computed in a similar manner to $\mathcal{J}^N(a_1,a_2,a_3)$. The objective function is first differentiated with respect to a_1, a_2, and a_3:

$$\frac{\partial}{\partial a_1}\mathcal{J}(a_1,a_2,a_3) = \int_0^1 -2(4y(1-y)-u(5,y;a_1,a_2,a_3))\frac{\partial u(5,y;a_1,a_2,a_3)}{\partial a_1}\mathrm{d}y \tag{4.16}$$

$$\frac{\partial}{\partial a_2}\mathcal{J}(a_1,a_2,a_3) = \int_0^1 -2(4y(1-y)-u(5,y;a_1,a_2,a_3))\frac{\partial u(5,y;a_1,a_2,a_3)}{\partial a_2}\mathrm{d}y \tag{4.17}$$

$$\frac{\partial}{\partial a_3}\mathcal{J}(a_1,a_2,a_3) = \int_0^1 -2(4y(1-y)-u(5,y;a_1,a_2,a_3))\frac{\partial u(5,y;a_1,a_2,a_3)}{\partial a_3}\mathrm{d}y. \tag{4.18}$$

The approximations u^N, $(\frac{\partial u}{\partial a_1})^N$, $(\frac{\partial u}{\partial a_2})^N$, and $(\frac{\partial u}{\partial a_3})^N$ are returned by the flow solver on triangular elements, so the coordinate transformation and resulting computations take place again. Once u^N and the sensitivity derivatives have been computed at the nine necessary points, the composite trapezoid rule is used to evaluate the integral. Thus,

$$\left(\frac{\partial}{\partial a_1}\mathcal{J}(a_1,a_2,a_3)\right)^N = h[-2(4y_1(1-y_1)-u^N(5,y_1;a_1,a_2,a_3))(\frac{\partial u}{\partial a_1})^N \ldots - 2(4y_9(1-y_9)-u^N(5,y_9;a_1,a_2,a_3))(\frac{\partial u}{\partial a_1})^N] \quad (4.19)$$

$$\left(\frac{\partial}{\partial a_2}\mathcal{J}(a_1,a_2,a_3)\right)^N = h[-2(4y_1(1-y_1)-u^N(5,y_1;a_1,a_2,a_3))(\frac{\partial u}{\partial a_2})^N \ldots - 2(4y_9(1-y_9)-u^N(5,y_9;a_1,a_2,a_3))(\frac{\partial u}{\partial a_2})^N] \quad (4.20)$$

$$\left(\frac{\partial}{\partial a_3}\mathcal{J}(a_1,a_2,a_3)\right)^N = h[-2(4y_1(1-y_1)-u^N(5,y_1;a_1,a_2,a_3))(\frac{\partial u}{\partial a_3})^N \ldots - 2(4y_9(1-y_9)-u^N(5,y_9;a_1,a_2,a_3))(\frac{\partial u}{\partial a_3})^N], \quad (4.21)$$

where $y_1 = 0.1$, $y_2 = 0.2$, $\ldots y_8 = 0.8$, $y_9 = 0.9$.

4.3.5 Trust Region Algorithm

A trust region algorithm is applied to the flow problem using each of the two gradient computation methods. A secant optimization method is used. Convergence criteria is either a gradient value or step value less than 10^{-11}. The initial trust region radius is 0.1, and a value of $\Delta a = 0.001$ is used for computation of the forward difference gradient approximation. Again, the objective function is very flat near the optimal point, so the function and gradient values are scaled by a factor of 10^5. Optimization is performed for several different cases, including cases where the flow and sensitivity equations are approximated using either two or three mesh refinements.

Chapter 5

Numerical Results and Discussion

5.1 ODE Example

5.1.1 Objective Function

Figure 5.1 is a snapshot of the objective function $\mathcal{J}^N(a)$ for each of the three error tolerance cases and $a \in [0.012, 0.01255]$. Figure 5.2 is an enlargement of the graph of $\mathcal{J}^N(a)$ in the same a region. As can be seen in Figure 5.1, the adaptive time-stepping solution algorithm has caused numerical noise on the order of 10^{-3} in the objective function at `rel_tol`$= 10^{-3}$. There is also a large discrepancy in the values of $\mathcal{J}^N(a)$ between the first case and the other two cases. Although the true minimum a_* for this problem is not known, based on a values computed every $\Delta a = 10^{-6}$, the minimum value of $\mathcal{J}^N(a) = 2.5 \times 10^{-9}$ occurs at $a_* \approx \frac{1}{82.45} \approx 0.012129$ for `rel_tol`$= 10^{-6}$. For `rel_tol`$= 10^{-3}$, the smallest value of $\mathcal{J}^N(a) = 2.1 \times 10^{-3}$ occurs at $a \approx 0.012206$. There is a noticeable difference between this a value and a_*. Although we do not have a true graph of $\mathcal{J}(a)$, our best approximation is $\mathcal{J}^N(a)$ at `rel_tol`$= 10^{-6}$. For `rel_tol`$= 10^{-6}$, $\mathcal{J}^N(0.012206) = 1.1 \times 10^{-6}$. This value is 1000 times larger than $\mathcal{J}^N(a_*)$ at the same tolerance level. Clearly, the weaker tolerance level returns an inferior result.

As is visible in Figure 5.2, numerical noise is evident for the second case on a scale of order 10^{-5}. The function looks relatively smooth near the minimum value of $a \approx 0.012181$. For `rel_tol`$= 10^{-6}$, $\mathcal{J}^N(0.012181) = 5.2 \times 10^{-7}$. Although the value of a is closer to a_* than in the case of the weakest tolerance, this value of a is still noticeably to the right of a_*, and $\mathcal{J}^N(a)$ is still 100 times larger than $\mathcal{J}^N(a_*)$. It is clear that in this example, tighter tolerance levels lead to more accurate function evaluations.

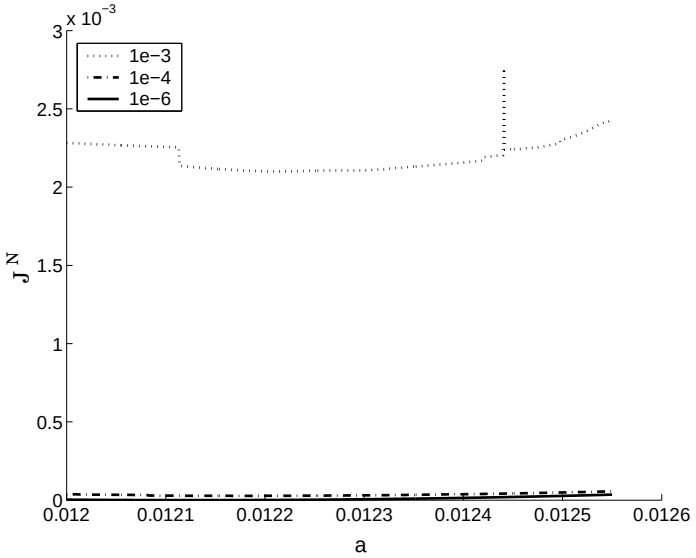

Figure 5.1: ODE Example: Objective Function

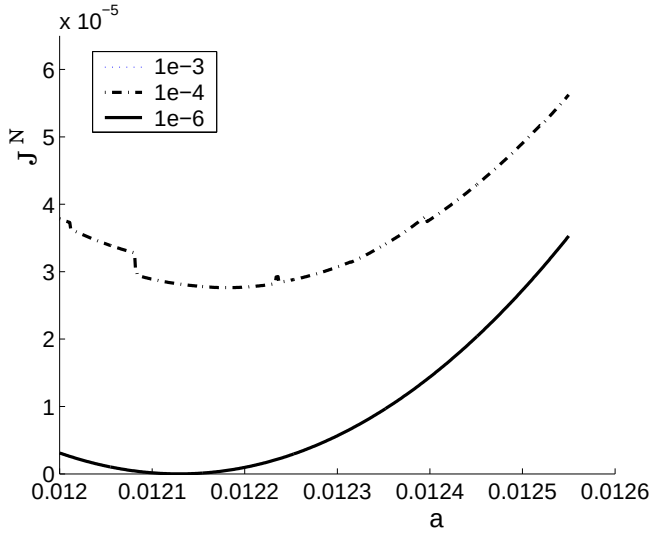

Figure 5.2: ODE Example: Amplified Objective Function

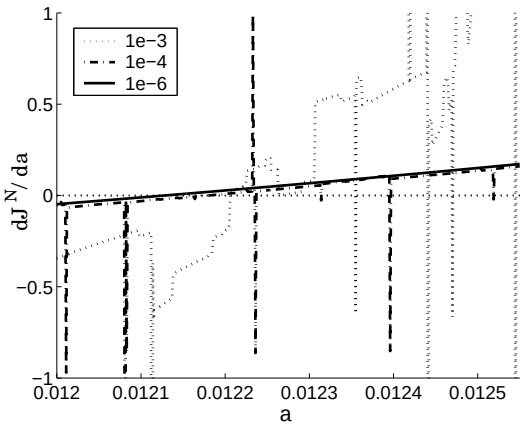

Figure 5.3: ODE Example: Finite Difference Gradients

5.1.2 Finite Difference Gradient

Figure 5.3 displays plots of the finite difference gradients of $\mathcal{J}^N(a)$ for each of the three tolerance levels. As is demonstrated in Figures 5.1 and 5.2, $\mathcal{J}^N(a)$ increases on each side of the minimum a value for all three tolerance levels. Thus, $\frac{d\mathcal{J}^N(a)}{da}$ should be positive to the right of a_* and negative to the left of a_*. However, the finite difference gradient does not behave as expected for the first two cases. The gradients for the weakest and medium tolerance levels are very (numerically) noisy. The gradient plots reflect more noise than the objective function plots. For case one, the gradient changes signs frequently. The gradient does change sign near the minimum $a \approx 0.012206$, but the gradient crosses zero at several other a values. The results for the medium tolerance level also demonstrate several points where $\frac{d\mathcal{J}^N(a)}{da}$ crosses zero. For the tightest tolerance level, the gradient only passes through zero at one point. The point is $a \approx 0.012128$, which is extremely close to a_*.

5.1.3 CSE Gradient

Figure 5.4 plots the CSE gradient approximations. For the weakest tolerance level, the gradient remains negative over the entire a range. For the medium tolerance level, the CSE gradient approximation crosses zero only once at $a \approx 0.012148$, which is relatively near a_*. For the tightest tolerance level, $(\frac{d\mathcal{J}(a)}{da})^N$ is equal to zero only once at $a = 0.012128$, which is even closer to a_*. For this problem, small errors in the computed value of a_* lead to non-periodic solutions.

Gradient based optimization algorithms will have difficulty with the finite difference gradients that cross zero in more than one place [27]. Often, algorithms will converge based on the size of the gradient. As the gradient reaches a value close enough to zero, the algorithm will stop searching for better objective function values. The finite difference gradient values

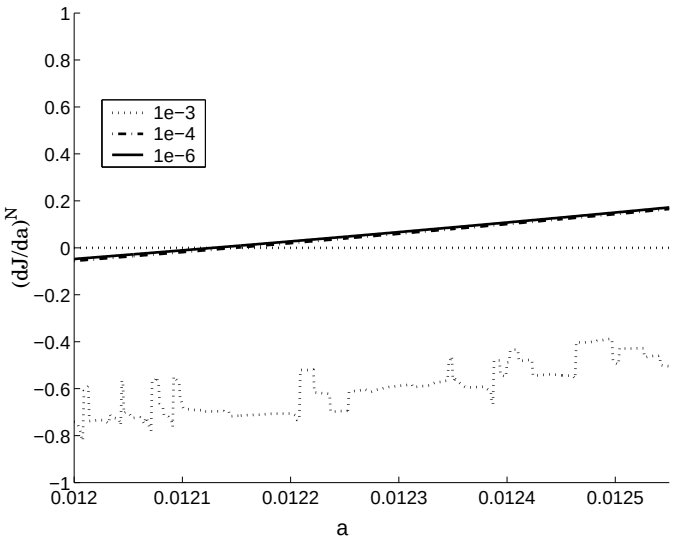

Figure 5.4: ODE Example: CSE Gradient Approximations

for the first and second cases could cause optimization algorithms to terminate at a values far distant from a_*. The points where $\frac{d\mathcal{J}^N(a)}{da}$ is equal to zero on these graphs are not even local minimums. They are solely the result of numerical noise created by the solution technique. Additionally, gradient based optimization algorithms use the sign of the gradient to determine which direction to search. Since $\frac{d\mathcal{J}^N(a)}{da}$ is both positive and negative on each side of a_* for the first and second cases, optimization algorithms using this gradient information could search in the wrong direction (i.e. the direction where $\mathcal{J}^N(a)$ is increasing rather than decreasing). This scenario could also lead to false convergence, as the algorithm might conclude that there are no decreasing values of $\mathcal{J}^N(a)$. At the tightest tolerance level, $\frac{d\mathcal{J}^N(a)}{da}$ is always the correct sign, and it only crosses zero at a single point near a_*. Thus, the finite difference gradient would likely cause an optimization algorithm to converge to a point very near a_* when the tightest tolerance is used.

For `rel_tol`$= 10^{-3}$, the CSE gradient approximation never crosses zero in our parameter range. Since $(\frac{d\mathcal{J}(a)}{da})^N$ is never close to zero, convergence of an optimization algorithm using this gradient is unlikely. However, for the second case, the CSE gradient approximation has the correct sign on each side of a_*, and only crosses zero once at $a \approx 0.012148$. This a value is close to the minimum value of a for `rel_tol`$= 10^{-4}$ and to a_*. Thus, an optimization algorithm using this gradient would likely stop at to a point near a_*. For the tightest tolerance, again the CSE gradient approximation has the correct sign on each side of a_*. Additionally, the gradient only crosses zero once at $a \approx 0.012129$, which is essentially equal to a_*. So, optimization algorithms using this gradient would terminate at a point near a_*.

5.2 Flow Example: Single Parameter

5.2.1 Objective Function

Figure 5.5 is a plot of $\mathcal{J}^N(a)$ over the entire a range for four adaptations. Recall that $\mathcal{J}(a)$ was constructed to have a global minimum at $a = 0$. The plot of $\mathcal{J}^N(a)$ reflects a single minimum near $a = 0$. Figures 5.6 through 5.9 are enlarged plots of $\mathcal{J}^N(a)$ for one, two, three, and four adaptations over a smaller a range. These figures demonstrate considerable numerical noise in the objective function for each iteration. The amount of noise in $\mathcal{J}^N(a)$ clearly decreases with each successive mesh refinement. However, even after multiple adaptations, $\mathcal{J}^N(a)$ is still not a smooth function.

Figures 5.10 through 5.13 are depictions of the meshes produced when the flow solver adapted the flow solution four times for $a = 0.25$. The mesh becomes visibly finer above the obstruction with each refinement. Since the flow is initially and eventually parabolic, large quadratic elements are sufficient to resolve the flow velocity and the essentially zero velocity sensitivity before and after the bump. The subsequent mesh refinements are the reason that numerical noise in the objective function decreases with each adaptation. Figures 5.14 through 5.17 are the flow profiles provided by the flow solver after each of the four adaptations. In these figures, it is evident that the flow solution becomes smoother with successive adaptations.

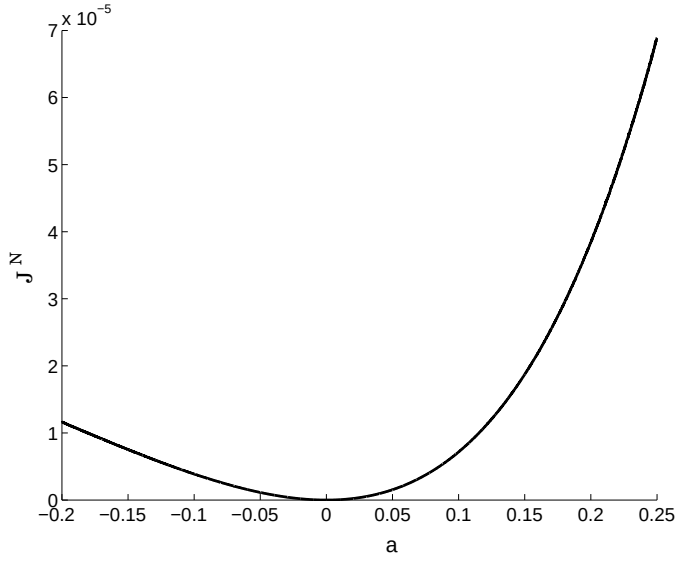

Figure 5.5: Single Parameter Flow Example: Objective Function After Four Adaptations

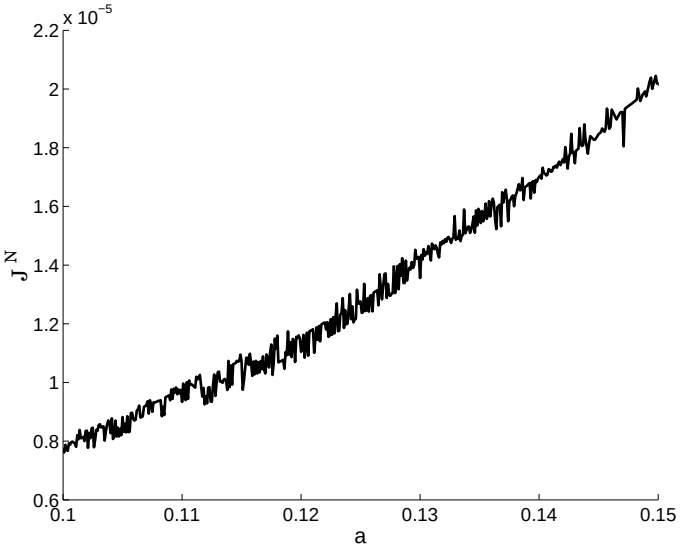

Figure 5.6: Single Parameter Flow Example: Amplified Objective Function After One Adaptation

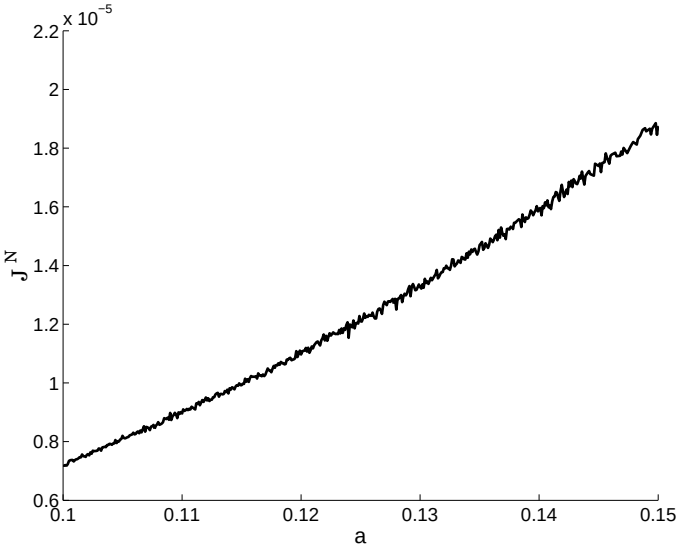

Figure 5.7: Single Parameter Flow Example: Amplified Objective Function After Two Adaptations

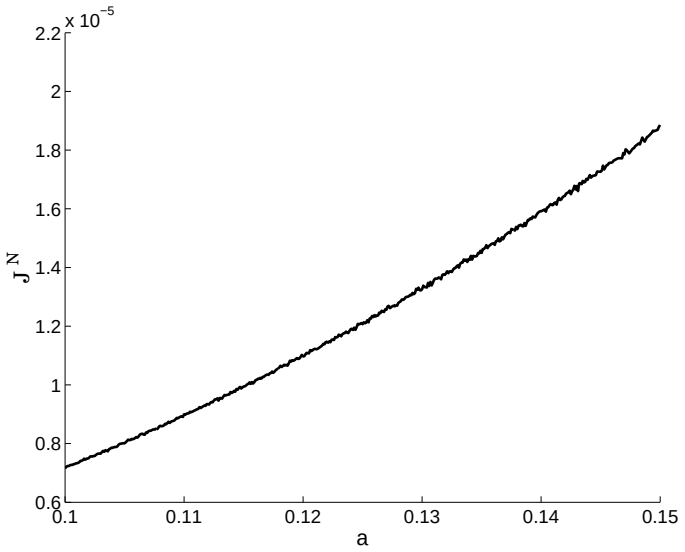

Figure 5.8: Single Parameter Flow Example: Amplified Objective Function After Three Adaptations

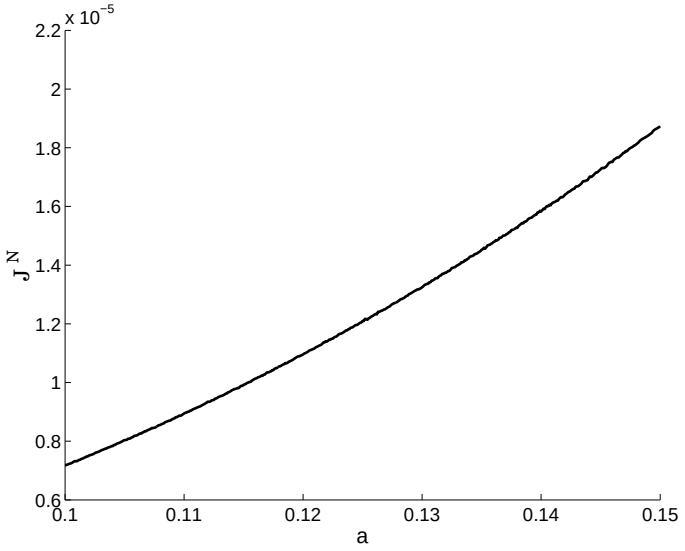

Figure 5.9: Single Parameter Flow Example: Amplified Objective Function After Four Adaptations

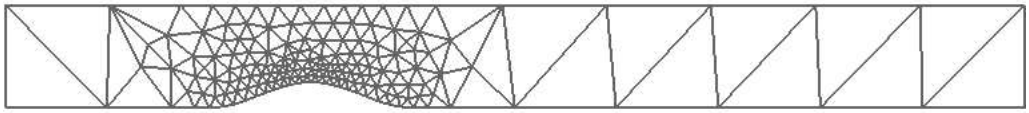

Figure 5.10: Single Parameter Flow Example: Mesh After One Adaptation, $a = 0.25$

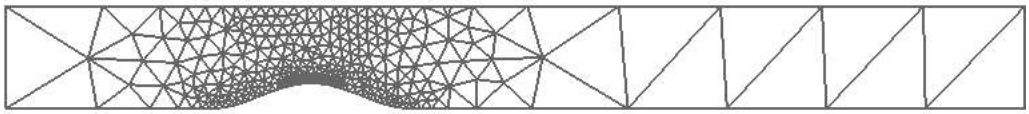

Figure 5.11: Single Parameter Flow Example: Mesh After Two Adaptations, $a = 0.25$

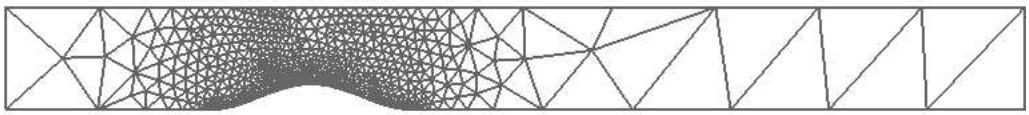

Figure 5.12: Single Parameter Flow Example: Mesh After Three Adaptations, $a = 0.25$

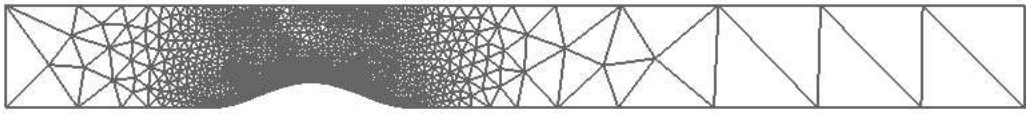

Figure 5.13: Single Parameter Flow Example: Mesh After Four Adaptations, $a = 0.25$

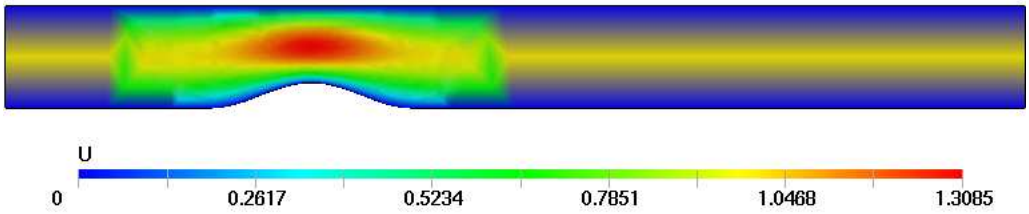

Figure 5.14: Single Parameter Flow Example: Flow Profile After One Adaptation, $a = 0.25$

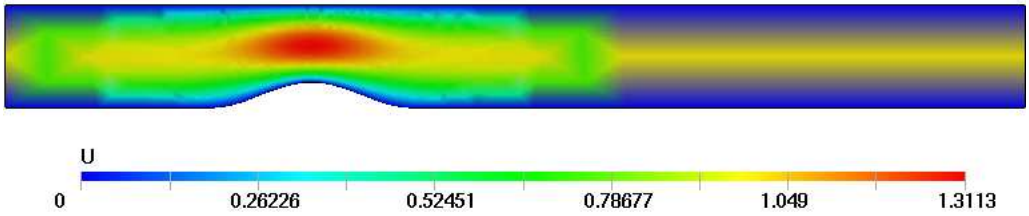

Figure 5.15: Single Parameter Flow Example: Flow Profile After Two Adaptations, $a = 0.25$

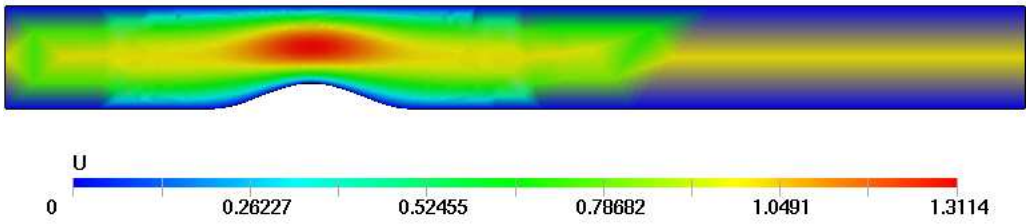

Figure 5.16: Single Parameter Flow Example: Flow Profile After Three Adaptations, $a = 0.25$

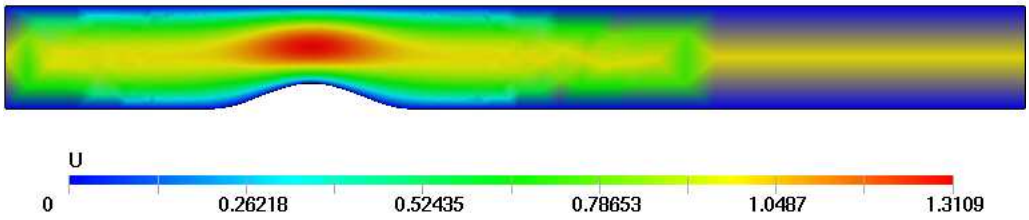

Figure 5.17: Single Parameter Flow Example: Flow Profile After Four Adaptations, $a = 0.25$

5.2.2 Finite Difference Gradients

Table 5.1 tabulates the wild values that result for $a = 0.2$ when finite difference methods are used to compute gradients for varying step sizes. At $a = 0.2$, the function is increasing, and thus the gradient is positive. For both the central difference and forward difference methods and each of the four adaptations, the sign of the gradient depends upon the step size Δa. The sign of the gradient determines the search direction in gradient based optimization algorithms. Thus, choosing a step size that is too large or too small could force an optimization algorithm to search in a direction away from the true minimum. Even if the chosen step size results in a gradient of the correct sign for one a value, the sign of the gradient could be incorrect for the next value of a. The magnitude of the gradient also varies widely with changes in Δa for both the forward difference and central difference schemes. Although the magnitude of the gradient does not factor heavily into the performance of most optimization algorithms, the variation of gradient magnitudes with respect to step size indicates that gradient calculations become less reliable as the step size is decreased. The finite difference method is unable to compute meaningful gradient values for the first two adaptations of this flow problem. The objective function approximation contains so much noise that computed derivatives are meaningless for even small step values.

Table 5.1: Single Parameter Flow Example: Discrete and CSE Gradients

Cycle	Step Size	Forward	Central	CSE
1	10^{-2}	2.289	1.144	0.0108
	10^{-3}	-0.004	-0.003	
	10^{-4}	-0.004	-0.003	
	10^{-5}	-0.025	-0.001	
2	10^{-2}	2.124	0.285	0.0294
	10^{-3}	18.35	-1.018	
	10^{-4}	-10.66	-0.208	
	10^{-5}	-59.55	-7.358	
3	10^{-2}	-0.127	0.005	0.2915
	10^{-3}	-0.456	1.706	
	10^{-4}	5.205	4.567	
	10^{-5}	14.22	-35.659	
4	10^{-2}	0.283	0.025	0.2971
	10^{-3}	0.120	0.033	
	10^{-4}	-2.701	-2.666	
	10^{-5}	17.47	-7.069	

5.2.3 CSE Gradient

Table 5.1 also tabulates the CSE gradient approximation for each adaptation at $a = 0.2$. It is worth noting that the CSE gradient returns a gradient value of the correct sign for each adaptation. $\mathcal{J}^N(a)$ increases as a increases, and the positive value of the CSE gradient approximation reflects this fact. Figures 5.18 through 5.21 are the plots of the CSE gradient approximations for each of the four adaptations. (Plots like these were not possible for the finite difference gradients at any step size due to the variations in sign and magnitude of the computed gradients). These plots demonstrate that $(\frac{d\mathcal{J}(a)}{da})^N$ is either equal to zero or very close to zero at many points after one, two, and three adaptations. Figures 5.22 and 5.23 reveal that the points where $(\frac{d\mathcal{J}(a)}{da})^N$ is incorrectly equal to zero are results of partial convergence. Figure 5.22 is a plot of $-2(4y(1-y)-u^N(5,y;0.25))$ after one adaptation, and Figure 5.23 is a plot of $(\frac{\partial u}{\partial a})^N(0.25)$ after one adaptation. The derivative $(\frac{d\mathcal{J}(a)}{da})^N(0.25)$ is the integral of the product of these two functions. Note that these plots are nearly symmetric and anti-symmetric about the line $y = 0.5$, respectively. Unresolved solutions can lead to cases where the interpolated trapezoidal integration points do not accurately capture the subtle variation in the velocity. Thus, the derivative $(\frac{d\mathcal{J}(a)}{da})^N(0.25)$ is incorrectly computed as zero. After four adaptations, there is more detail in the finite element solution, and the effects of partial convergence are no longer evident. Figures 5.24 and 5.25 reflect the increased accuracy at $a = 0.25$ when four adaptations are used. After four adaptations, the derivative is only equal to zero at one point, as we would expect from Figure 5.5.

If an optimization algorithm evaluated the gradient at one of these points, premature convergence could result. However, for most points in all four adaptations, $(\frac{d\mathcal{J}(a)}{da})^N$ is the correct sign. Thus, unless $(\frac{d\mathcal{J}(a)}{da})^N = 0$, the CSE gradient approximation will direct optimization algorithms to search in the correct direction. Additionally, for each of the four adaptations, the CSE gradient approximation is equal to zero at a values very close to $a = 0$, which is the true minimum. Although there are points in the first three adaptations that could result in convergence problems, the CSE gradient will result in better optimization outcomes than the finite difference gradients. There is no question over which step size to use, or how the step size will affect search directions. For the CSE gradient, the search direction will be correct for most a points even after only one adaptation. There are many opportunities for the finite difference gradients to point optimization algorithms in the wrong search direction.

Figures 5.26 through 5.29 are representations of the computed results for $(\frac{\partial u}{\partial a})^N$ at $a = 0.25$. These pictures demonstrate that the flow values are most sensitive to a directly above the obstruction.

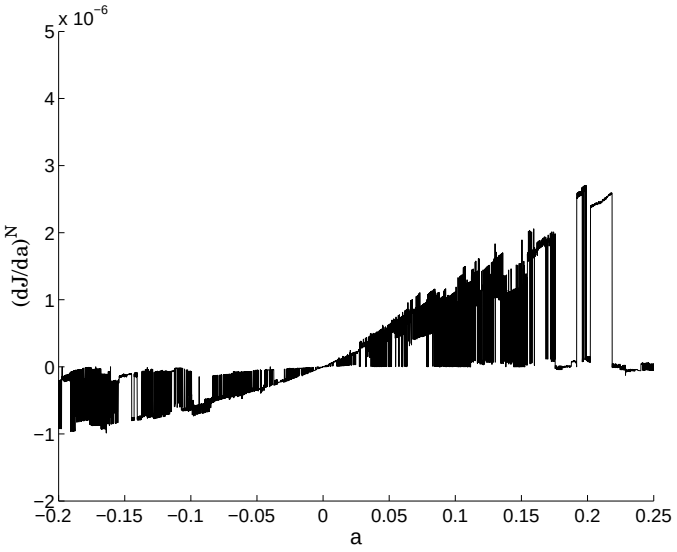

Figure 5.18: Single Parameter Flow Example: CSE Gradient Approximation After One Adaptation

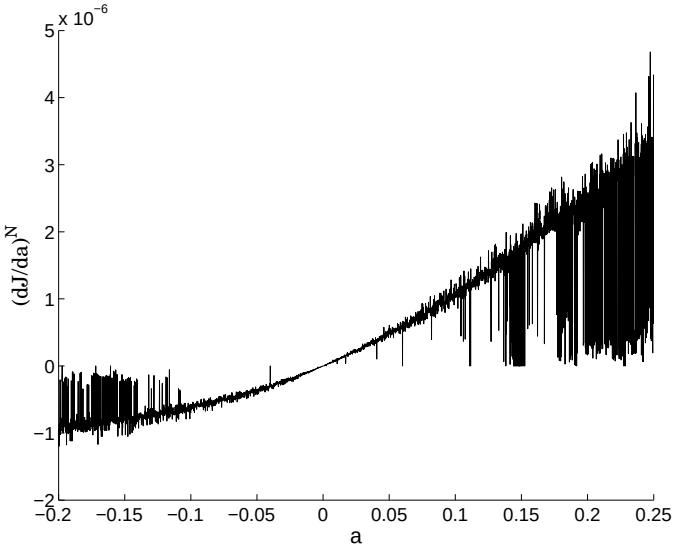

Figure 5.19: Single Parameter Flow Example: CSE Gradient Approximation After Two Adaptations

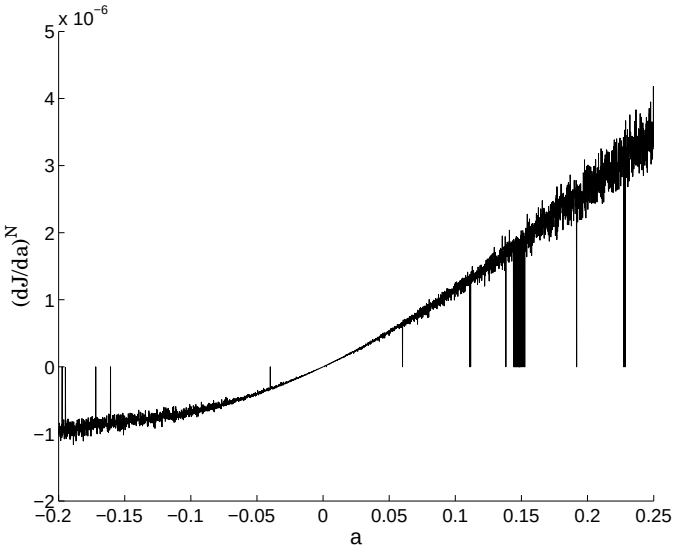

Figure 5.20: Single Parameter Flow Example: CSE Gradient Approximation After Three Adaptations

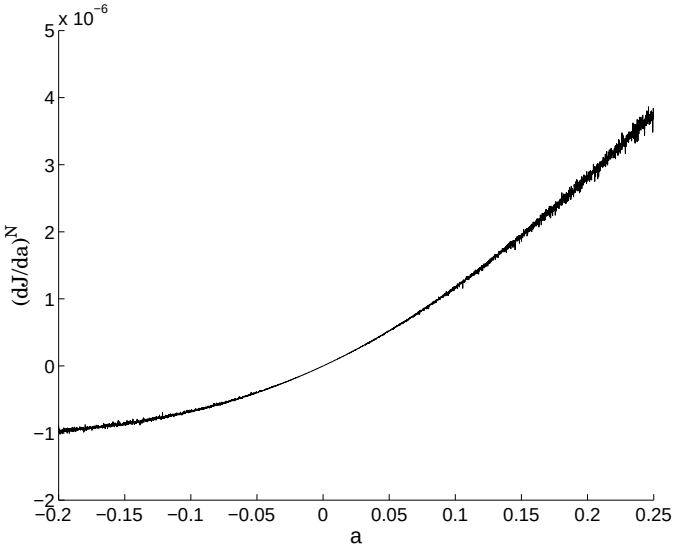

Figure 5.21: Single Parameter Flow Example: CSE Gradient Approximation After Four Adaptations

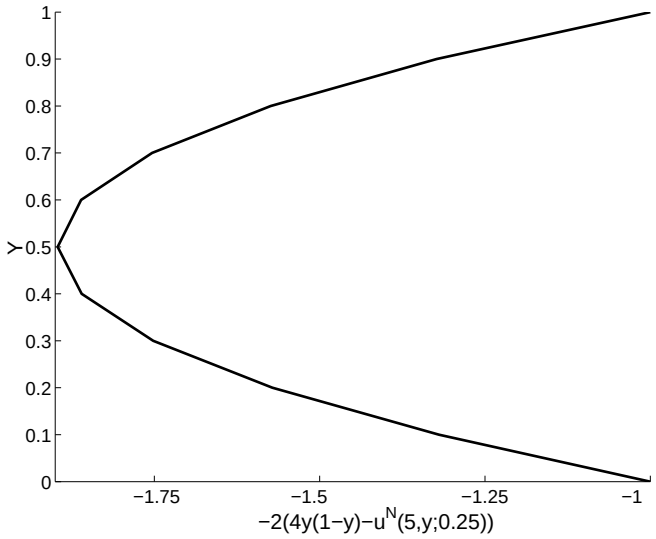

Figure 5.22: Single Parameter Flow Example: Difference of Velocity Profiles After One Adaptation, $a = 0.25$

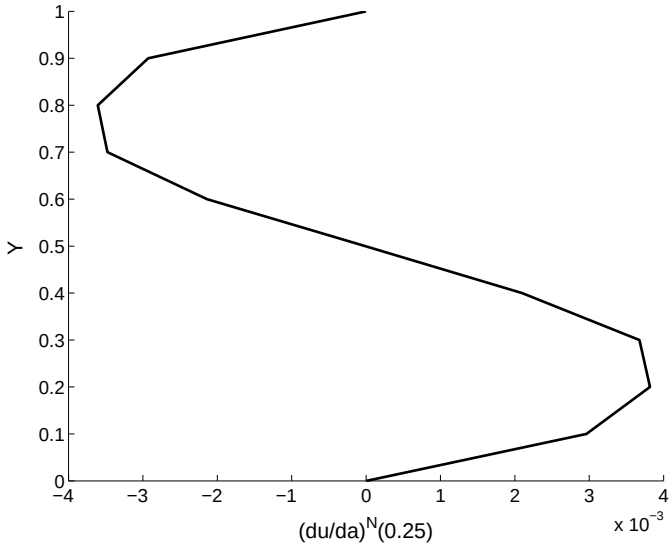

Figure 5.23: Single Parameter Flow Example: $(\frac{\partial u}{\partial a})^N$ After One Adaptation, $a = 0.25$

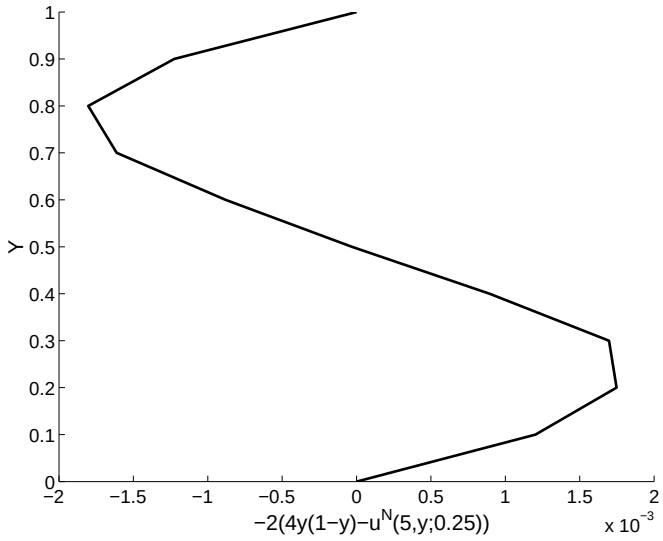

Figure 5.24: Single Parameter Flow Example: Difference of Velocity Profiles After Four Adaptations, $a = 0.25$

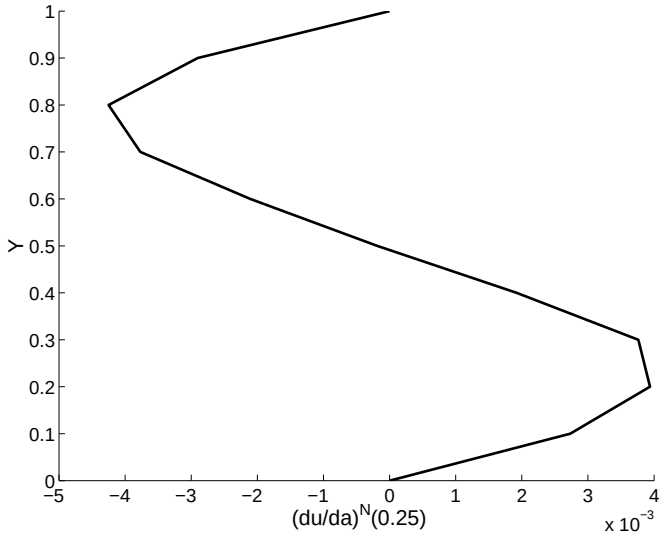

Figure 5.25: Single Parameter Flow Example: $(\frac{\partial u}{\partial a})^N$ After Four Adaptations, $a = 0.25$

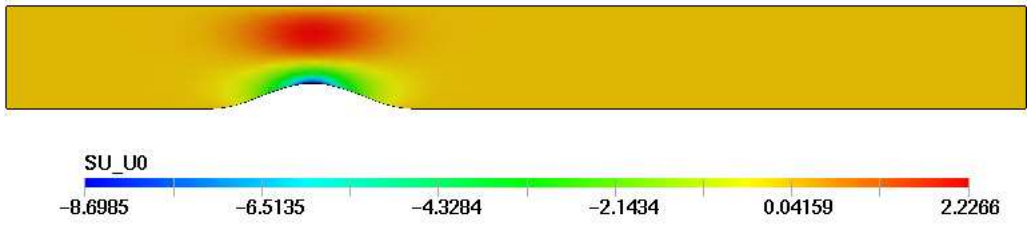

Figure 5.26: Single Parameter Flow Example: Flow Sensitivity Profile After One Adaptation, $a = 0.25$

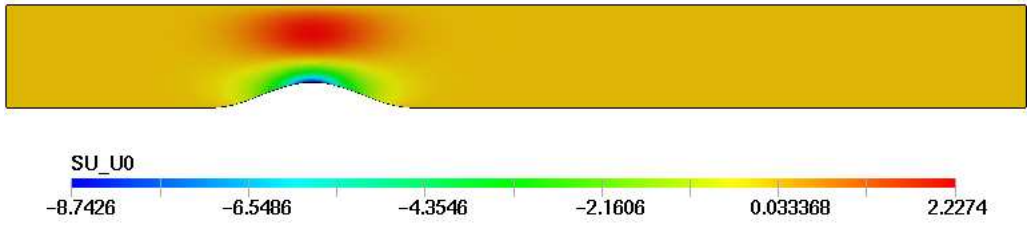

Figure 5.27: Single Parameter Flow Example: Flow Sensitivity Profile After Two Adaptations, $a = 0.25$

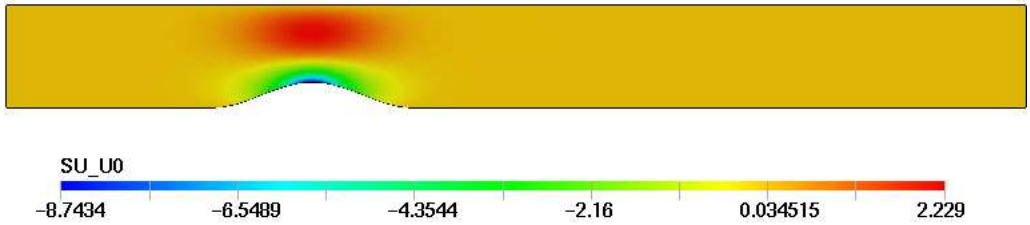

Figure 5.28: Single Parameter Flow Example: Flow Sensitivity Profile After Three Adaptations, $a = 0.25$

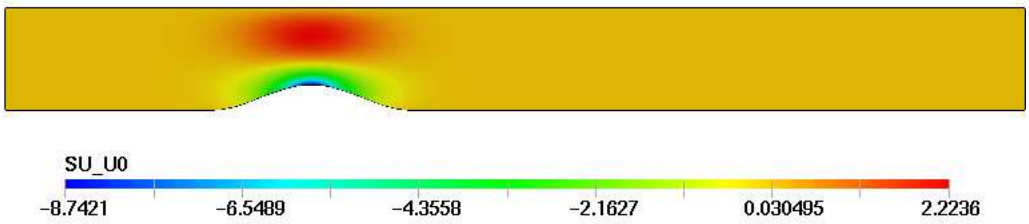

Figure 5.29: Single Parameter Flow Example: Flow Sensitivity Profile After Four Adaptations, $a = 0.25$

5.2.4 Optimization Results

A trust region optimization algorithm is applied to $\mathcal{J}^N(a)$ after three adaptations using gradients computed with the forward difference method and the CSE method. Table 5.2 is a summary of optimization results for $a_0 = -0.1$. For both gradients, the trust region algorithm halted due to step size shrinking below tolerance levels. For this starting point, the CSE gradient outperforms the forward difference gradient. When the CSE gradient is used, the algorithm returns a final value of $\mathcal{J}^N = 6.3 \times 10^{-16}$ at $a_*^C = 1.00 \times 10^{-8}$. By comparison, the finite difference gradient results in $\mathcal{J}^N = 9.6 \times 10^{-6}$ at $a_*^D = -0.00492$. The CSE gradient approximation returns $\mathcal{J}^N$ and a values that are much closer to the optimum values of $\mathcal{J}^N = 0$ at $a = 0$ than the finite difference gradient approximation. Additionally, the trust region algorithm combined with the CSE gradient only requires 18 gradient evaluations prior to termination. When the finite difference gradient is supplied to the optimization algorithm, 26 gradient evaluations are necessary.

Table 5.2: Single Parameter Flow Example: Optimization Results, $a_0 = -0.1$

Discrete

Iter.	a^D	$\mathcal{J}^N$	$\frac{\partial}{\partial a}\mathcal{J}^N$
0	-1.00×10^{-1}	3.3×10^{-3}	-1.2×10^{-2}
1	-8.81×10^{-2}	1.9×10^{-3}	-8.4×10^{-2}
2	-4.27×10^{-3}	1.5×10^{-5}	6.9×10^{-4}
3	-4.95×10^{-3}	9.6×10^{-6}	-6.5×10^{-5}
4	-4.92×10^{-3}	9.6×10^{-6}	-1.2×10^{-5}

CSE

Iter.	a^C	$\mathcal{J}^N$	$(\frac{\partial}{\partial a}\mathcal{J})^N$
0	-1.00×10^{-1}	3.7×10^{-3}	-6.7×10^{-2}
1	-3.26×10^{-2}	4.5×10^{-4}	-2.7×10^{-2}
2	1.22×10^{-2}	7.0×10^{-5}	1.2×10^{-2}
3	-1.36×10^{-3}	8.7×10^{-7}	-1.3×10^{-2}
4	-2.04×10^{-5}	2.0×10^{-10}	-1.9×10^{-5}
5	2.47×10^{-8}	6.5×10^{-16}	1.4×10^{-8}
6	1.00×10^{-8}	6.3×10^{-16}	1.6×10^{-8}

Table 5.3 is a comparison of optimization results for $a_0 = 0.15$. Again, the CSE gradient returns better results than the forward difference gradient. When the CSE gradient is used, $\mathcal{J}^N = 4.8 \times 10^{-16}$ at $a_*^C = -1.21 \times 10^{-8}$. When the forward difference gradient is used, $\mathcal{J}^N = 5.0 \times 10^{-8}$ at $a_*^D = -1.03 \times 10^{-2}$. Again, the CSE gradient results in $\mathcal{J}^N$ and a values that are much closer to the optimum values. When the trust region algorithm uses the CSE

Table 5.3: Single Parameter Flow Example: Optimization Results, $a_0 = 0.15$

Discrete

Iter.	a^D	$\mathcal{J}^N$	$\frac{\partial}{\partial a}\mathcal{J}^N$
0	1.50×10^{-1}	1.3×10^{-2}	1.0×10^{-1}
1	5.00×10^{-2}	1.5×10^{-3}	3.9×10^{-2}
2	-1.24×10^{-2}	2.4×10^{-6}	-5.9×10^{-3}
3	-1.04×10^{-2}	5.4×10^{-8}	-4.5×10^{-3}
4	-1.03×10^{-2}	5.0×10^{-8}	-4.3×10^{-3}

CSE

Iter.	a^C	$\mathcal{J}^N$	$(\frac{\partial}{\partial a}\mathcal{J})^N$
0	1.50×10^{-1}	1.2×10^{-2}	1.8×10^{-1}
1	5.00×10^{-2}	1.3×10^{-3}	5.4×10^{-2}
2	8.87×10^{-3}	3.7×10^{-5}	8.4×10^{-3}
3	1.20×10^{-3}	6.6×10^{-7}	1.1×10^{-3}
4	4.20×10^{-5}	8.0×10^{-10}	3.8×10^{-5}
5	6.85×10^{-7}	2.3×10^{-13}	6.7×10^{-7}
6	-4.95×10^{-8}	1.3×10^{-15}	-3.6×10^{-8}
7	-1.21×10^{-8}	4.8×10^{-16}	-6.4×10^{-9}

gradient, only 18 gradient evaluations are necessary. However, the forward difference method necessitates 35 gradient evaluations.

In Tables 5.4 and 5.5, the trust region algorithm uses both gradients to optimize the flow problem for starting points $a_0 = 0.1186$ and $a_0 = -0.07515$. As the tables indicate, the CSE gradient outperforms the finite difference gradient for both starting points. For the two starting points, the CSE gradient requires 21 and 15 gradient evaluations, respectively, while the forward difference gradients requires 35 gradient evaluations for both of the starting points. However, the most striking result that is evident from these tables is that for these two starting points, the trust region algorithm combined with the forward difference gradient results in termination at a values farther away from zero than the a_0 values. When the CSE gradient is used, the final a_*^C values are noticeable improvements over the a_0 values. But, when the finite difference gradient is used, the starting point $a_0 = 0.1186$ leads to termination at $a_*^D = 0.11861$. Although the computed value of $\mathcal{J}^N(a_*^D)$ is less than $\mathcal{J}^N(a_0)$, we know that $\mathcal{J}(a_*^D)$ is actually greater than $\mathcal{J}(a_0)$, since $\mathcal{J}(a)$ is an increasing function for $a > 0$. For $a_0 = -0.07515$, the forward difference gradient leads the trust region algorithm to terminate at $a_*^D = -0.07525$. This result represents an even more significant step in the direction away from the true optimum.

There are two factors that have caused the trust region algorithm to converge to points

that are further from optimum than the starting point. The first factor is that the forward difference gradient has the wrong sign at each of the a_0 values. Thus, the trust region algorithm automatically searches in the wrong direction for a smaller value of $\mathcal{J}^N$. The second factor that causes the trust region algorithm to accept the step in the wrong direction is that there is a value $|a| > |a_0|$ such that $\mathcal{J}^N(a) < \mathcal{J}^N(a_0)$. Although this should not be the case, the level of numerical noise in $\mathcal{J}^N(a)$ makes it possible.

The number of gradient evaluations required by the trust region algorithm is also an issue. When finite difference gradient information is needed, the objective function must be evaluated twice in order to compute the gradient. Gradient information only requires one function evaluation for the CSE method. In this example, the flow solver is the most expensive part of the objective function evaluation process. Since the flow solver must be used for every objective function evaluation, computing the finite difference gradient at each step of the trust region algorithm adds a significant amount of time to each iteration. The flow solver takes more time to solve a flow problem with sensitivities than one without sensitivities. However, the solution of two problems without sensitivity data takes longer than the solution of one problem with sensitivity calculations. So, every gradient evaluation required by the trust region algorithm takes longer for the finite difference gradient than for the CSE gradient. Additionally, the trust region algorithm requires fewer gradient evaluations for the CSE gradient approximation than for the finite difference gradient for each point we considered. Thus, two factors lead the trust region algorithm to converge in less time when the CSE gradient is used. The amount of time needed to compute gradient information is less, and the number of gradient evaluations is also lower.

Table 5.4: Single Parameter Flow Example: Optimization Results, $a_0 = 0.1186$

Discrete

Iter.	a^D	$\mathcal{J}^N$	$\frac{\partial}{\partial a}\mathcal{J}^N$
0	1.19×10^{-1}	6.9×10^{-3}	1.8×10^{0}
1	1.19×10^{-1}	6.7×10^{-3}	5.1×10^{0}
2	1.19×10^{-1}	6.4×10^{-3}	-4.3×10^{0}

CSE

Iter.	a^C	$\mathcal{J}^N$	$(\frac{\partial}{\partial a}\mathcal{J})^N$
0	1.19×10^{-1}	7.8×10^{-3}	1.4×10^{-1}
1	1.86×10^{-2}	1.6×10^{-4}	1.7×10^{-2}
2	4.69×10^{-3}	1.0×10^{-5}	4.4×10^{-3}
3	3.74×10^{-5}	6.5×10^{-10}	3.5×10^{-5}
4	2.13×10^{-7}	2.1×10^{-14}	1.9×10^{-7}
5	3.88×10^{-9}	2.0×10^{-15}	4.8×10^{-8}
6	-3.10×10^{-8}	9.0×10^{-16}	-2.4×10^{-8}
7	-1.93×10^{-8}	7.0×10^{-16}	-1.6×10^{-8}
8	-1.34×10^{-8}	4.8×10^{-16}	-6.4×10^{-9}

Table 5.5: Single Parameter Flow Example: Optimization Results, $a_0 = -0.07515$

Discrete

Iter.	a^D	$\mathcal{J}^N$	$\frac{\partial}{\partial a}\mathcal{J}^N$
0	-7.52×10^{-2}	2.0×10^{-3}	8.8×10^{-1}
1	-7.52×10^{-2}	1.8×10^{-3}	-1.9×10^{0}
2	-7.52×10^{-2}	1.8×10^{-3}	-5.4×10^{-1}

CSE

Iter.	a^C	$\mathcal{J}^N$	$(\frac{\partial}{\partial a}\mathcal{J})^N$
0	-7.52×10^{-2}	2.4×10^{-3}	-5.8×10^{-2}
1	-1.67×10^{-2}	1.3×10^{-4}	-1.5×10^{-2}
2	3.17×10^{-3}	4.6×10^{-6}	2.9×10^{-3}
3	-9.63×10^{-5}	4.4×10^{-9}	-9.1×10^{-5}
4	2.53×10^{-6}	3.0×10^{-12}	2.3×10^{-6}
5	5.73×10^{-8}	2.4×10^{-15}	5.7×10^{-8}
6	-5.01×10^{-9}	4.8×10^{-16}	1.9×10^{-9}

5.3 Flow Example: Multiple Parameters

The multiple parameter flow example presents several issues that were not a factor for the single parameter flow problem. First, plots of the objective function are not possible, since the objective function depends upon three parameters. Likewise, the gradient vector is not easily represented in graphical form. Since the general shapes of $\mathcal{J}(a_1, a_2, a_3)$ and $\mathcal{J}^N(a_1, a_2, a_3)$ are not known, it is difficult to determine whether a gradient approximation points in an increasing direction or a decreasing direction. Even if the computed gradient does point in an increasing direction, it is not known if this is the direction of steepest increase that the true gradient would indicate. Although analysis and comparison of gradient signs was very enlightening for the single parameter flow example, the correct signs of the multiple parameter problem gradients are unknown. Thus, the analysis and comparison of the two different gradient approximation methods is restricted to the performance of the trust region algorithm when combined with approximate gradient information.

The trust region algorithm is first combined with the forward difference gradient. The mesh is refined twice for this example, and $\vec{a_0} = (0.02, 0.02, 0.02)$. Table 5.6 summarizes the results. The trust region algorithm requires 40 gradient evaluations prior to termination. There is a noticeable improvement in $\mathcal{J}^N(a_1, a_2, a_3)$ values, and the (a_1, a_2, a_3) values at termination are all closer to zero than the $\vec{a_0}$ values. Figures 5.30 and 5.31 are channel geometry for $\vec{a_0}$ and the termination point $(a_1, a_2, a_3) = (0.0257, 0.00924, -0.000763)$. The optimal channel geometry is a flat bottom, and the channel is closer to being flat at the termination point. The obstruction is lower than for $\vec{a_0}$ and flatter towards the outlet.

Figure 5.30: Multiple Parameter Flow Example: Channel Domain, $\vec{a}_0 = (0.02, 0.02, 0.02)$

Figure 5.31: Multiple Parameter Flow Example: Channel Domain, $\vec{a}_* = (0.0257, 0.00924, -0.000763)$

Table 5.6: Multiple Parameter Flow Example: Optimization Results, $\vec{a}_0 = (0.02, 0.02, 0.02)$

Discrete			
Iter.	a_1, a_2, a_3	$\mathcal{J}^N$	$\frac{\partial \mathcal{J}^N}{\partial a_1}$, $\frac{\partial \mathcal{J}^N}{\partial a_2}$, $\frac{\partial \mathcal{J}^N}{\partial a_3}$
0	2.00×10^{-2}	9.8×10^{-2}	-5.8×10^{0}
	2.00×10^{-2}		1.4×10^{1}
	2.00×10^{-2}		2.7×10^{1}
1	2.47×10^{-2}	5.0×10^{-3}	-3.3×10^{0}
	8.42×10^{-3}		-3.6×10^{0}
	-1.65×10^{-3}		-4.4×10^{0}
2	2.57×10^{-2}	3.7×10^{-5}	6.4×10^{0}
	9.25×10^{-3}		6.7×10^{0}
	-7.63×10^{-4}		2.0×10^{-1}
3	2.57×10^{-2}	3.0×10^{-5}	3.8×10^{-1}
	9.24×10^{-3}		6.6×10^{0}
	-7.63×10^{-4}		7.3×10^{0}
4	2.57×10^{-2}	1.8×10^{-5}	6.4×10^{0}
	9.24×10^{-3}		7.0×10^{0}
	-7.63×10^{-4}		3.8×10^{-1}

Figure 5.32: Multiple Parameter Flow Example: Channel Domain, $\vec{a}_0 = (-0.02, 0.02, 0.02)$

Figure 5.33: Multiple Parameter Flow Example: Channel Domain, $\vec{a}_* = (0.0251, 0.00760, -0.00793)$

The trust region algorithm is combined with the CSE gradient for another initial obstruction shape corresponding to $\vec{a_0} = (-0.02, 0.02, 0.02)$. The mesh is refined twice for this example, and results are summarized in Table 5.7. The trust region algorithm required 43 gradient evaluations prior to termination. In this case, there is also a marked reduction in $\mathcal{J}^N(a_1, a_2, a_3)$. However, as Figures 5.32 and 5.33 demonstrate, the channel geometry does not drastically flatten. The largest part of the obstruction is moved towards the inlet rather than reduced.

Both gradient approximation methods are combined with the trust region algorithm for $\vec{a_0} = (0.0015, -0.0015, 0.002)$. Table 5.8 summarizes the results. For both gradient approximations, the mesh is refined twice. The CSE gradient results in a smaller objective function value at termination than the forward difference gradient. Also, each $\vec{a}$ component at termination of the CSE run is closer to zero than the $\vec{a}$ components at termination of the forward difference run. Thus, the CSE gradient led the trust region algorithm to terminate at a point closer to the true minimum than the finite difference gradient. The trust region algorithm requires 31 gradient evaluations when the CSE gradient is used and 40 gradient evaluations when the forward difference gradient is used. This trend occurred for the single parameter flow problem too–optimization requires more gradient evaluations for the discrete gradient approximation. The flow solver requires more time to solve a three parameter problem than a one parameter problem. Also, for the three parameter problem, a finite difference gradient approximation requires at least four function evaluations. Thus, the flow solver must execute four times for each of the trust region algorithm's gradient evaluations. As in the single parameter case, the flow solver is faster when solving a problem without sensitivity data. But, the extra function evaluations for forward difference gradient computation force

Table 5.7: Multiple Parameter Flow Example: Optimization Results, $\vec{a}_0 = (-0.02, 0.02, 0.02)$

	CSE		
Iter.	a_1,a_2,a_3	$\mathcal{J}^N$	$(\frac{\partial \mathcal{J}}{\partial a_1})^N$, $(\frac{\partial \mathcal{J}}{\partial a_2})^N$, $(\frac{\partial \mathcal{J}}{\partial a_3})^N$
0	-2.00×10^{-2}	8.5×10^{-2}	8.5×10^{-1}
	2.00×10^{-2}		3.2×10^{0}
	2.00×10^{-2}		6.2×10^{0}
1	-2.60×10^{-2}	4.7×10^{-2}	-6.6×10^{-1}
	-2.94×10^{-3}		-1.2×10^{0}
	-2.40×10^{-2}		-2.3×10^{0}
2	2.36×10^{-2}	1.4×10^{-2}	-3.5×10^{-1}
	2.30×10^{-3}		-8.3×10^{-1}
	-2.04×10^{-2}		-1.4×10^{0}
3	2.60×10^{-2}	5.1×10^{-4}	-6.7×10^{-2}
	9.27×10^{-3}		-2.0×10^{-1}
	-1.03×10^{-2}		-3.8×10^{-1}
4	2.61×10^{-2}	1.8×10^{-4}	3.9×10^{-2}
	9.92×10^{-3}		9.8×10^{-2}
	-8.93×10^{-3}		1.6×10^{-1}
5	2.53×10^{-2}	1.6×10^{-5}	1.9×10^{-3}
	8.66×10^{-3}		5.2×10^{-3}
	-8.61×10^{-3}		5.8×10^{-3}
6	2.51×10^{-2}	1.3×10^{-5}	-9.2×10^{-3}
	7.59×10^{-3}		-3.9×10^{-2}
	-7.95×10^{-3}		-5.6×10^{-2}
7	2.51×10^{-2}	1.3×10^{-5}	-6.6×10^{-3}
	7.60×10^{-3}		-2.1×10^{-2}
	-7.93×10^{-3}		-3.7×10^{-2}
8	2.51×10^{-2}	3.7×10^{-6}	-4.0×10^{-3}
	7.60×10^{-3}		-2.2×10^{-2}
	-7.93×10^{-3}		-2.2×10^{-2}

Table 5.8: Multiple Parameter Flow Example: Optimization Results, $\vec{a}_0 = (0.0015, -0.0015, 0.002)$

Discrete

Iter.	a_1, a_2, a_3	$\mathcal{J}^N$	$\frac{\partial \mathcal{J}^N}{\partial a_1}$, $\frac{\partial \mathcal{J}^N}{\partial a_2}$, $\frac{\partial \mathcal{J}^N}{\partial a_3}$
0	1.50×10^{-3}	2.0×10^{-4}	-1.9×10^{-1}
	-1.50×10^{-3}		1.2×10^{-1}
	2.00×10^{-3}		3.7×10^{-1}
1	2.19×10^{-3}	5.9×10^{-7}	-5.9×10^{-4}
	-1.93×10^{-3}		1.9×10^{-3}
	6.67×10^{-4}		1.1×10^{-1}
2	2.19×10^{-3}	3.2×10^{-7}	-2.1×10^{-4}
	-1.93×10^{-3}		2.1×10^{-3}
	6.65×10^{-4}		9.0×10^{-2}
3	2.19×10^{-3}	1.1×10^{-7}	-1.1×10^{-4}
	-1.93×10^{-3}		1.4×10^{-2}
	6.64×10^{-4}		1.1×10^{-1}

CSE

Iter.	a_1,a_2,a_3	$\mathcal{J}^N$	$(\frac{\partial \mathcal{J}}{\partial a_1})^N$, $(\frac{\partial \mathcal{J}}{\partial a_2})^N$, $(\frac{\partial \mathcal{J}}{\partial a_3})^N$
0	1.50×10^{-3}	2.1×10^{-4}	3.3×10^{-2}
	-1.50×10^{-3}		1.2×10^{-1}
	2.00×10^{-3}		2.7×10^{-1}
1	1.32×10^{-3}	3.7×10^{-6}	-4.6×10^{-3}
	-2.13×10^{-3}		-1.6×10^{-2}
	5.79×10^{-4}		-3.5×10^{-2}
2	1.70×10^{-3}	2.3×10^{-8}	-6.6×10^{-5}
	-1.70×10^{-3}		-3.1×10^{-4}
	5.39×10^{-4}		-6.2×10^{-4}
3	1.69×10^{-3}	1.2×10^{-8}	2.0×10^{-4}
	-1.67×10^{-3}		6.6×10^{-4}
	5.30×10^{-4}		1.5×10^{-3}

the trust region algorithm to run much longer when the forward difference gradient is used.

Finally, the impact of the number of mesh adaptations is analyzed. The CSE gradient is supplied to the trust region algorithm after two and three mesh adaptations for $\vec{a_0} = (-0.0015, -0.0015, -0.002)$. Tables 5.9 and 5.10 summarize the results. As the tables illustrate, $\vec{a}_{2*} = (-0.00034, 0.000292, -0.000088)$ and $\vec{a}_{3*} = (-0.000389, 0.000202, -0.0000407)$ Figures 5.34 through 5.39 represent the meshes and flow profiles for $\vec{a_0}$ after one, two, and three adaptations. (Note that the abrupt change in velocity at the inlet and after the obstruction is an artifact of the visualization software, which assumes piecewise linear functions.) The flow profile is smoother after the final adaptation, and $\mathcal{J}^N$ is smaller at termination for three mesh adaptations rather than two. The termination point $\vec{a_*}$ is also closer to zero after three adaptations than the termination point after two adaptations. The best approximation that we can feasibly compute to $\mathcal{J}(\vec{a})$ is $\mathcal{J}^N(\vec{a})$ after four mesh adaptations. It is interesting to note that $\mathcal{J}^N(\vec{a}_{2*}) = 2.1 \times 10^{-9}$ and $\mathcal{J}^N(\vec{a}_{3*}) = 4.3 \times 10^{-12}$.

The trust region algorithm requires 32 gradient evaluations for two mesh refinements and 35 gradient evaluations for three mesh refinements. Despite the similar number of gradient evaluations, the trust region algorithm takes more than twice as long to run for three adaptations as it takes for two adaptations. In this example, as in any practical optimization problem, the goal of extra accuracy must be weighed against the cost of longer function evaluations.

Table 5.9: Multiple Parameter Flow Example: Optimization Results After Two Refinements, $\vec{a}_0 = (-0.0015, -0.0015, -0.002)$

CSE: Two Mesh Refinements

Iter.	a_1,a_2,a_3	$\mathcal{J}^N$	$(\frac{\partial \mathcal{J}}{\partial a_1})^N$, $(\frac{\partial \mathcal{J}}{\partial a_2})^N$, $(\frac{\partial \mathcal{J}}{\partial a_3})^N$
0	-1.50×10^{-3}	7.4×10^{-4}	-6.9×10^{-2}
	-1.50×10^{-3}		-2.3×10^{-1}
	-2.00×10^{-3}		-5.0×10^{-1}
1	-1.11×10^{-3}	3.4×10^{-5}	1.5×10^{-2}
	-2.09×10^{-4}		5.3×10^{-2}
	8.20×10^{-4}		1.2×10^{-1}
2	-3.09×10^{-4}	5.0×10^{-7}	-1.7×10^{-3}
	2.62×10^{-4}		-6.0×10^{-3}
	-1.55×10^{-4}		-1.3×10^{-3}
3	-3.33×10^{-4}	7.4×10^{-9}	-2.2×10^{-4}
	2.94×10^{-4}		-7.6×10^{-4}
	-9.99×10^{-5}		-1.6×10^{-3}
4	-3.33×10^{-4}	1.0×10^{-9}	4.2×10^{-6}
	3.09×10^{-4}		1.0×10^{-5}
	-9.72×10^{-5}		-8.9×10^{-6}
5	-3.35×10^{-4}	6.3×10^{-10}	-1.2×10^{-5}
	3.03×10^{-4}		-4.6×10^{-5}
	-9.42×10^{-5}		-1.2×10^{-4}
6	-3.40×10^{-4}	2.5×10^{-10}	2.5×10^{-5}
	2.92×10^{-4}		8.1×10^{-5}
	-8.80×10^{-5}		1.7×10^{-4}

Table 5.10: Multiple Parameter Flow Example: Optimization Results After Three Refinements, $\vec{a}_0 = (-0.0015, -0.0015, -0.002)$

CSE: Three Mesh Refinements

Iter.	a_1,a_2,a_3	$\mathcal{J}^N$	$(\frac{\partial \mathcal{J}}{\partial a_1})^N$, $(\frac{\partial \mathcal{J}}{\partial a_2})^N$, $(\frac{\partial \mathcal{J}}{\partial a_3})^N$
0	-1.50×10^{-3}	8.1×10^{-4}	-7.0×10^{-2}
	-1.50×10^{-3}		-2.5×10^{-1}
	-2.00×10^{-3}		-5.6×10^{-1}
1	-1.15×10^{-3}	3.8×10^{-5}	1.5×10^{-2}
	-2.60×10^{-4}		5.5×10^{-2}
	8.46×10^{-4}		1.3×10^{-1}
2	-5.90×10^{-4}	1.9×10^{-7}	-1.1×10^{-3}
	-2.01×10^{-4}		-3.9×10^{-3}
	1.20×10^{-4}		-8.7×10^{-3}
3	-5.69×10^{-4}	1.3×10^{-8}	-9.5×10^{-6}
	-1.19×10^{-4}		1.8×10^{-5}
	1.22×10^{-4}		1.9×10^{-4}
4	-5.40×10^{-4}	8.2×10^{-9}	-2.8×10^{-5}
	-6.70×10^{-5}		-5.8×10^{-5}
	9.60×10^{-5}		-1.9×10^{-5}
5	-4.18×10^{-4}	1.3×10^{-9}	7.3×10^{-5}
	1.53×10^{-4}		2.7×10^{-4}
	-1.25×10^{-5}		6.4×10^{-4}
6	-3.93×10^{-4}	1.0×10^{-10}	2.4×10^{-5}
	1.95×10^{-4}		8.7×10^{-5}
	-3.65×10^{-5}		2.0×10^{-4}
7	-3.92×10^{-4}	1.1×10^{-11}	-8.6×10^{-8}
	1.96×10^{-4}		9.5×10^{-7}
	-3.76×10^{-5}		5.7×10^{-6}
8	-3.88×10^{-4}	4.5×10^{-12}	-1.6×10^{-6}
	2.04×10^{-4}		-5.9×10^{-6}
	-4.17×10^{-5}		-1.5×10^{-5}
9	-3.89×10^{-4}	3.5×10^{-12}	-3.7×10^{-7}
	2.02×10^{-4}		-1.4×10^{-6}
	-4.07×10^{-5}		-3.4×10^{-6}
10	-3.89×10^{-4}	3.4×10^{-12}	3.2×10^{-7}
	2.02×10^{-4}		9.6×10^{-7}
	-4.07×10^{-5}		1.7×10^{-6}
11	-3.89×10^{-4}	3.1×10^{-12}	5.7×10^{-7}
	2.02×10^{-4}		1.9×10^{-6}
	-4.07×10^{-5}		3.9×10^{-6}

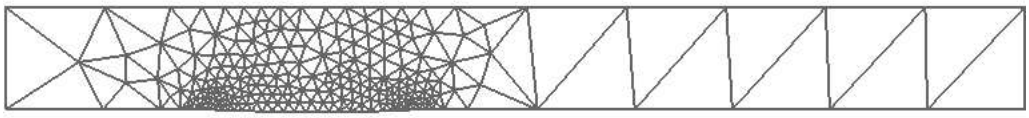

Figure 5.34: Multiple Parameter Flow Example: Mesh After One Adaptation, $\vec{a} = (-0.0015, -0.0015, -0.002)$

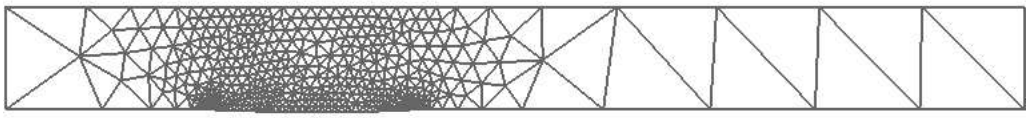

Figure 5.35: Multiple Parameter Flow Example: Mesh After Two Adaptations, $\vec{a} = (-0.0015, -0.0015, -0.002)$

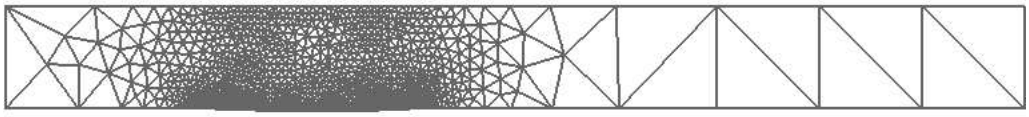

Figure 5.36: Multiple Parameter Flow Example: Mesh After Three Adaptations, $\vec{a} = (-0.0015, -0.0015, -0.002)$

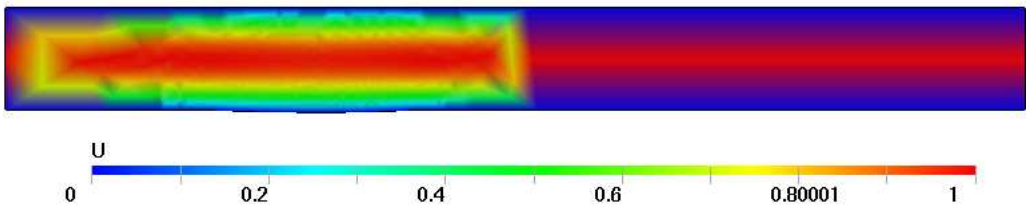

Figure 5.37: Multiple Parameter Flow Example: Flow Profile After One Adaptation, $\vec{a} = (-0.0015, -0.0015, -0.002)$

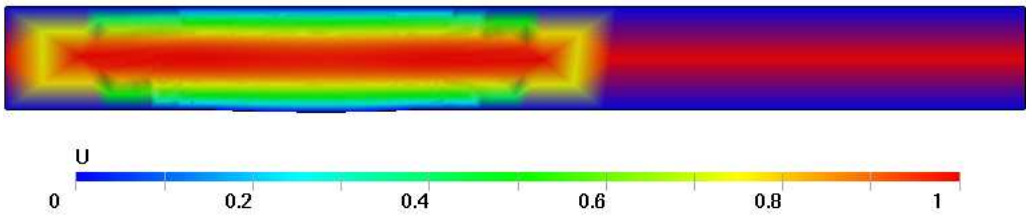

Figure 5.38: Multiple Parameter Flow Example: Flow Profile After Two Adaptations, $\vec{a} = (-0.0015, -0.0015, -0.002)$

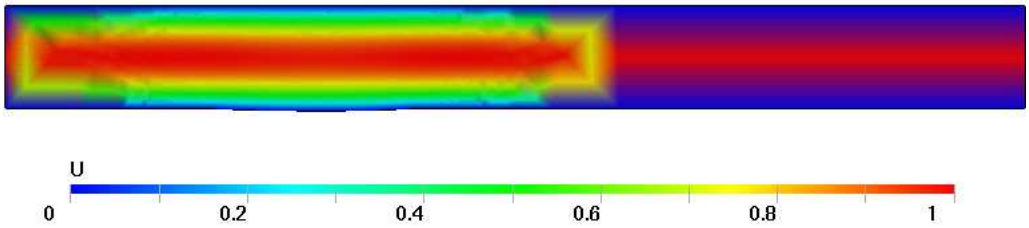

Figure 5.39: Multiple Parameter Flow Example: Flow Profile After Three Adaptations, $\vec{a} = (-0.0015, -0.0015, -0.002)$

Chapter 6

Conclusions and Future Plans

The results from all three examples indicate that CSE gradient approximations can compete very favorably with discrete gradient approximations in the presence of numerical noise. In the ODE and single parameter flow examples, graphs of CSE and finite difference gradients demonstrated that both gradients had undesirable attributes at high levels of noise. At moderate noise levels, the CSE gradient approximation was more useful in an optimization context than the finite difference gradient. For both of the flow examples, a trust region algorithm coupled with the CSE gradient approximation outperforms the finite difference gradient approximation. Also, optimization using the CSE gradient approximation terminates faster than optimization combined with the finite difference gradient approximation. The difference in optimization time is more evident in the higher dimensional case. The results from these numerical investigations indicate that more analysis of CSE methods is warranted.

At this time, only a limited number of studies have investiaged the convergence properties of trust region algorithms combined with CSE gradient approximations. Since the CSE gradient approximation is not computed directly from $\mathcal{J}^N$, the question of convergence in the face of inconsistent gradient information is important. Convergence of the trust region algorithm in the presence of numerical noise in the objective function and gradient has been proven under certain assumptions [15, 6, 22]. Our future goal is to establish that a trust region algorithm coupled with CSE gradient approximations will converge to within a certain accuracy of the true minimum. The accuracy of the converged result will depend upon the amount of numerical noise in the objective function and gradient.

Bibliography

[1] E. Anderson and M. Ferris. A direct search algorithm for optimization with noisy function evaluations. *SIAM J. Optim.*, 11(3):837–857, 2001.

[2] M. S. Bazaraa, H. D. Sherali, and C. M. Shetty. *Nonlinear Programming: Theory and Algorithms*. Wiley-Interscience Series in Discrete Mathematics and Optimization. John Wiley & Sons, Inc., New York, 1993.

[3] J. Borggaard and J. Burns. A sensitivity equation approach to optimal design of nozzles. In *Proc. 5th AIAA/USAF/NASA/ISSMO Symposium on Multidisciplinary Analysis and Optimization*, pages 232–241, 1994.

[4] J. Borggaard and J. Burns. A sensitivity equation approach to shape optimization in fluid flows. In M. Gunzburger, editor, *Flow Control*, volume 68 of *Proceedings of the IMA*. Springer-Verlag, 1995.

[5] J. Borggaard and J. Burns. Asymptotically consistent gradients in optimal design. In N. Alexandrov and M. Hussaini, editors, *Multidisciplinary Design Optimization: State of the Art*, pages 303–314, Philadelphia, PA, 1997. SIAM.

[6] J. Borggaard and J. Burns. A PDE sensitivity equation method for optimal aerodynamic design. *J. Comput. Phys.*, 136(2):366–384, 1997.

[7] J. Borggaard, J. Burns, E. Cliff, and M. Gunzburger. Sensitivity calculations for a 2D, inviscid, supersonic forebody problem. In H. T. Banks, R. Fabiano, and K. Ito, editors, *Identification and Control of Systems Governed by Partial Differential Equations*, pages 14–24, Philadelphia, PA, 1993. SIAM.

[8] J. Borggaard, J.-F. Hétu, and D. Pelletier. Parametric uncertainty analysis in a phase change model. In *Proc. 9th AIAA/NASA/USAF/ISSMO Symposium on Multidisciplinary Analysis and Optimization*, 2002. AIAA Paper 2002-5601.

[9] J. Borggaard and D. Pelletier. Optimal shape design in forced convection using adaptive finite elements. In *Proc. 36th AIAA Aerospace Sciences Meeting and Exhibit*, 1998. AIAA Paper 98-0908.

[10] J. Borggaard, D. Pelletier, and K. Vugrin. On sensitivity analysis for problems with numerical noise. In *Proc. 9th AIAA/NASA/USAF/ISSMO Symposium on Multidisciplinary Analysis and Optimization*, 2002. AIAA Paper 2002-5553.

[11] D. M. Bortz and C. T. Kelley. The simplex gradient and noisy optimization problems. In J. Borggaard, J. Burns, E. Cliff, and S. Schreck, editors, *Computational Methods for Optimal Design and Control*, pages 77–90. Birkhäuser, 1998.

[12] J. Burkardt, M. Gunzburger, and J. Peterson. Discretization of cost sensitivities in shape optimization. In *Computation and Control, IV*, volume 20 of *Progr. Systems Control Theory*. Birkhäuser, 1995.

[13] J. Burkardt, M. Gunzburger, and J. Peterson. Insensitive functionals, inconsistent gradients, spurious minima, and regularized functionals in flow optimization problems. *International Journal on Computational Fluid Dynamics*, 16(3):171–185, 2002.

[14] J. Burman and B. Gebart. Influence from numerical noise in the objective function for flow design optimization. *Internat. J. Numer. Methods Heat Fluid Flow*, 11(1):6–19, 2001.

[15] R. G. Carter. Numerical optimization in Hilbert space using inexact function and gradient evaluations. Technical Report 89-45, ICASE, 1989.

[16] R. G. Carter. On the global convergence of trust-region algorithms using inexact gradient information. *SIAM J. Numer. Anal.*, 28(1):251–265, 1991.

[17] R. G. Carter. Numerical experience with a class of algorithms for nonlinear optimization using inexact function and gradient information. *SIAM J. Sci. Comput.*, 14(2):368–388, 1993.

[18] A. Conn, K. Scheinberg, and P. Toint. On the convergence of derivative-free methods for unconstrained optimization. In M. D. Buhmann and A. Iserles, editors, *Approximation Theory and Optimization*, pages 83–108. Cambridge University Press, 1997.

[19] A. R. Conn, N. I. M. Gould, and P. L. Toint. *Trust-Region Methods*. MPS-SIAM Series on Optimization. SIAM, Philadelphia, 2000.

[20] M. C. Delfour and J.-P. Zolesio. Shape sensitivity analysis via MinMax differentiability. *SIAM J. Control Optim.*, 26(4), July 1988.

[21] J. E. Dennis Jr. and R. B. Schnabel. *Numerical Methods for Unconstrained Optimization and Nonlinear Equations*. Prentice Hall, Englewood Cliffs, New Jersey, 1983.

[22] U. Felgenhauer. Algorithmic stability analysis for certain trust region methods. In A. Fiacco, editor, *Mathematical Programming with Data Perturbations*, volume 195 of *Lecture Notes in Pure and Applied Math*. M. Dekker, 1998.

[23] W. Gautschi. *Numerical Analysis: An Introduction*. Birkhäuser, Boston, 1997.

[24] P. E. Gill, W. M. Murray, M. A. Saunders, and M. H. Wright. Computing forward-difference intervals for numerical optimization. *SIAM J. Sci. Comput.*, 4(2):310–321, June 1983.

[25] A. A. Giunta, V. Balbanov, D. Haim, B. Grossman, W. H. Mason, L. T. Watson, and R. T. Haftka. Multidisciplinary optimisation of a supersonic transport using design of experiments theory and response surface modelling. *Aeronautical Journal*, 101(1008):347–356, 1997.

[26] T. Glad and A. Goldstein. Optimization of functions whose values are subject to small errors. *BIT*, 17:160–169, 1977.

[27] M. S. Gockenbach and W. W. Symes. Adaptive simulation, the adjoint state method, and optimization. In *Proc. First Sandia Workshop on Large-Scale PDE-Constrained Optimization*. to appear.

[28] A. G. Godfrey, W. M. Eppard, and E. M. Cliff. Using sensitivity equations for chemically reacting flows. In *7th AIAA/USAF/NASA/ISSMO Symposium on Multidisciplinary Analysis and Optimization*, St. Louis, MO, September 2-4 1998.

[29] A. Griewank and G. F. Corliss, editors. *Automatic Differentiation of Algorithms: Theory, Implementation, and Application*. SIAM, Philadelphia, PA, 1991.

[30] G. J. W. Hou, J. L. Chen, and J. S. Sheen. Computational method for optimization of structural shapes. *AIAA Journal*, 24(6):1005–1012, 1986.

[31] G. J.-W. Hou, A. C. Taylor, and V. M. Korivi. Discrete shape sensitivity equations for aerodynamic problems. In *Proc. 27th AIAA/SAE/ASME/ASEE Joint Propulsion Conference*, 1991. AIAA Paper 91-2259.

[32] J. Iott, R. T. Haftka, and H. A. Adelman. Selecting step sizes in sensitivity analysis by finite differences. Technical Report NASA TM-86382, National Aeronautics and Space Administration, August 1985.

[33] K. Ito, H. Tran, and J. Scroggs. Mathematical issues in optimal design of a vapor transport reactor. In M. Gunzburger, editor, *Flow Control*, volume 68 of *Proceedings of the IMA*. Springer-Verlag, 1995.

[34] J. Kiefer and J. Wolfowitz. Stochastic estimation of the maximum of a regression function. *Annals of Mathematical Statistics*, 23:462–466, 1952.

[35] R. K. Miller and A. N. Michel. *Ordinary Differential Equations*. Academic Press, 1982.

[36] B. Mohammadi and O. Pironneau. *Applied Shape Optimization for Fluids*. Numerical Mathematics and Scientific Computation. Clarendon Press, Oxford, 2001.

[37] R. Narducci, B. Grossman, and R. Haftka. Design sensitivity algorithms for an inverse design problem involving a shock wave. *Inverse Problems in Engineering*, 2:49–83, 1995.

[38] L. Piegl and W. Tiller. *The NURBS Book*. Springer-Verlag, 1997.

[39] O. Pironneau. On optimum profiles in Stokes flow. *J. Fluid Mech.*, 59:117–128, 1973.

[40] O. Pironneau. On optimum design in fluid mechanics. *J. Fluid Mech.*, 60:97–110, 1974.

[41] G. D. Smith. *Numerical Solution of Partial Differential Equations: Finite Difference Methods*. Oxford Applied Mathematics and Computing Science Series. Clarendon Press, New York, 3rd edition, 1985.

[42] J. Sokolowski and J. P. Zolesio. *Introduction to Shape Optimization: Shape Sensitivity Analysis*, volume 16 of *Springer Series in Computational Mathematics*. Springer-Verlag, Berlin, 1991.

[43] L. Stanley. Shape sensitivities for optimal design: A case study on the use of continuous sensitivity equation methods. In D. Gao et al., editor, *Nonsmooth/Nonconvex Mechanics*. Kluwer Academic Publishers, 2001.

[44] A. C. Taylor III, G. W. Hou, and V. M. Korivi. A methodology for determining aerodynamic sensitivity derivatives with respect to variation of geometric shape. In *Proc. AIAA/ASME/ASCE/AHS/ASC 32nd Structures, Structural Dynamics, and Materials Conference*, Baltimore, MD, April 1991. AIAA paper 91-1101.

[45] A. C. Taylor III, G. W. Hou, and V. M. Korivi. Sensitivity analysis, approximate analysis and design optimization for internal and external viscous flows. In *Proc. AIAA Aircraft Design Systems and Operations Meeting*, 1991. AIAA paper 91-3083.

[46] P. L. Toint. Global convergence of a class of trust region algorithms for nonconvex minimization in hilbert space. *IMA Journal of Numerical Analysis*, 8(2):231–252, 1988.

[47] V. Torczon. On the convergence of the multidirectional search algorithm. *SIAM J. Optim.*, 1(1):123–145, 1991.

[48] V. Torczon. On the convergence of pattern search algorithms. *SIAM J. Optim.*, 7(1):1–25, 1997.

[49] É. Turgeon, D. Pelletier, and J. Borggaard. A general continuous sensitivity equation formulation for complex flows. In *Proc. 8th AIAA/NASA/USAF/ISSMO Symposium on Multidisciplinary Analysis and Optimization*, 2000. AIAA Paper 2000-4732.

[50] É. Turgeon, D. Pelletier, and J. Borggaard. Application of a sensitivity equation method to the $\kappa - \epsilon$ model of turbulence. In *Proc. 15th AIAA Computational Fluid Dynamics Conference*, 2001. AIAA Paper 2001–3000.

Vita

Kay Ellen White Vugrin was born in Lubbock, Texas to Ken and Barbara White on November 16, 1976. She graduated with Honors from Lubbock High School in 1995. She earned a Bachelor's Degree in Industrial Engineering and graduated Summa Cum Laude from Texas Tech University in 2000. Kay was awarded a National Science Foundation Graduate Research Fellowship to support her graduate studies. This thesis is the final requirement for her Master's Degree in Mathematics from Virginia Polytechnic Institute and State University.