2015

Shape Optimization for Drag Minimization Using the Navier-Stokes Equation

Chukwudi Paul Chukwudozie
Louisiana State University and Agricultural and Mechanical College

SHAPE OPTIMIZATION FOR DRAG MINIMIZATION USING THE NAVIER-STOKES EQUATION

A Thesis

Submitted to the Graduate Faculty of the
Louisiana State University and
Agricultural and Mechanical College
in partial fulfillment of the
requirements for the degree of
Master of Science

in

The Department of Mathematics

by
Chukwudi P. Chukwudozie
B. Eng., Federal University of Technology Minna, 2006
M.S., Louisiana State University, 2011
December 2015

Acknowledgments

I want to start by acknowledging the support of my family, Paulina and Chibuikem, my parents and siblings. I am eternally indebted to them for all their love and understanding.

I wish to extend my deepest gratitude to my supervisor, Dr. Shawn Walker. In so many ways, this project would not have been possible without him. His numerical linear algebra class was timely in my graduate school life while my interest in practical application of optimization, especially with regards to solving shape optimization problems, was piqued by his shape optimization class. My constant discussions with him during the period of this project further refined my theoretical optimization knowledge and guided me along the right path for numerical implementation of the drag minimization problem.

I also want to thank my committee members, Dr. Blaise Bourdin and Mayank Tyagi for graciously accepting to serve in the committee. In particular, I would like to express my appreciation to Dr. Blaise Bourdin for urging me to actualize this MS degree and for giving me the permission to embark on this project.

Numerical solutions of all partial differential equations in this project were implemented using FEniCS finite element library.

Table of Contents

List of Tables

List of Figures

List of Nomenclature

Symbols

σ	Stress tensor
$\vec{u}$	Fluid velocity
p	Fluid pressure
$\varepsilon(\vec{u})$	Strain rate
$\vec{z}$	Adjoint velocity
r	Adjoint pressure
$\vec{e}_x$	Basis vector in x direction
$\vec{v}$	Descent direction
$\vec{v}_{ex}$	Regularized descent direction
J	Fluid drag
$\mathcal{L}$	Drag Lagrangian
C	Geometric constraint
λ	Lagrange multiplier
A_o	Area of shape
$\vec{q}, \vec{\phi}, w$	Test functions
Ω	Computational domain
$\partial\Omega$	External boundaries of Ω
$\vec{n}$	Normal to boundary
Ω_b	Domain of obstacle
Γ_s	Obstacle boundary
Γ_+	Object boundary
Γ_-	Top boundary of Ω

Γ_{in}	Inlet boundary of Ω
Γ_{out}	Outlet boundary of Ω
I	Identity matrix in 2D
α	Step length for line search method
ϕ	Merit function
ρ	Penalty parameter in merit function

Abstract

Fluid drag is a force that opposes relative motion between fluid layers or between solids and surrounding fluids. For a stationary solid in a moving fluid, it is the amount of force necessary to keep the object stationary in the moving fluid. In addition to fluid and flow conditions, pressure drag on a solid object is dependent on the size and shape of the object. The aim of this project is to compute the shape of a stationary 2D object of size 3.5 m^2 that minimizes drag for different Reynolds numbers. We solve the problem in the context of shape optimization, making use of shape sensitivity analysis. The state variables are fluid pressure and velocity ($\vec{u}$ and p) modeled by the Navier-Stokes equation with cost function given by the fluid drag which depends on the state variables. The geometric constraint is removed by constructing a Lagrangian function. Subsequent application of shape sensitivity analysis on the Lagrangian generates the shape derivative and gradient. Our optimization routine uses a variational form of the sequential quadratic programming (SQP) method with the Hessian replaced by a variational form for the shape gradient. The numerical implementation is done in Python while the open source finite element package, FEniCS, is used to solve all the partial differential equations. Remeshing of the computational domain to improve mesh quality is carried out with the open source 2D mesh generator, Triangle. Final shapes for low Reynolds numbers ($Re \leq 1.0$) resemble an american football while shapes for moderate to high Reynolds numbers are ($Re \leq 200$) more streamlined in the tail end of the object than at the front.

Chapter 1
Introduction

Fluid mechanics is an important field of engineering that finds applications in various industries including aeronautics, automotive, manufacturing and chemical. Analysis of fluid flow facilitates engineering design and optimization for efficient utilization of energy and resources. One such application is in shape optimization in which design involves finding the optimal shape that minimizes fluid drag around a body. Knowledge from solution of this problem has been used to design shapes of airfoils and cars. Fluid drag is a force experienced by solids in contact with fluids. It resists relative motion between the solid and fluid and acts in the direction opposite to motion. It depends on flow conditions, solid shape and size and surface roughness. An inevitable consequence of drag is kinetic energy dissipation of the moving fluid or solid, leading to inefficiency in the use of energy. By finding the shape of airplanes and cars that minimizes drag, energy can be conserved and used to do other things like moving at faster speeds.

Mathematically, shape optimization involves a cost functional to be minimized, expressed in terms of state variables and defined over the physical domain of the problem or its boundaries. The state variables usually satisfy a set of equations, partial differential equations (PDE) for example. Thus, shape optimization problems belong to the the general class of PDE constrained optimization in which the control parameter is the object shape defined by its boundaries. The object whose shape is to be changed can be represented either implicitly or explicitly. Implicit representation identifies object boundaries as the level sets of some predefined functions while explicit representation uses a parameterization of the boundaries. In addition to equations that describe the state variables, the optimal shape sometimes is required to satisfy some geometric constraints like volume or shape requirement. This invariably introduces an extra level of complexity.

Mathematical analysis for the solution of shape optimization falls under shape differential calculus (Hadamard 1908; Pironneau 2012; Pironneau 1973; Pironneau 1973; Delfour and

Zolésio 2001; Delfour and Zolésio 2001; Zolésio 1992; Walker 2015). At the core of it is finding the directions along which the shape boundaries are moved to deform it towards the optimal profile. This direction is provided by the shape gradient which measures the sensitivity of the cost function with respect to small perturbations in the obstacle's shape. Thereafter, the boundaries are moved either by manually moving the coordinates of the boundary nodes (i.e. by updating the parameterization) or by advecting the phase field function representing the shape. Shape derivatives can be effectively combined with finite element methods to yield a method for computing optimal shapes. However, the corresponding strong form equations derived from weak form analysis can be solved using other numerical methods.

Chapter 2
Shape Sensitivity

2.1 Review of Basic Shape Calculus

We review the basic concepts and formulas in sensitivity analysis for shape optimization problems. Detailed information can be found in Haslinger et al. (2003), Zolésio (1992), Delfour and Zolésio (2001). Sensitivity analysis in shape optimization involves computations of derivatives of state, adjoint variables and cost functionals with respect to changes in object shape (Walker 2015). In this theory, the continuum mechanics approach is used and the object is considered as a collection of material particles changing position with time t, during deformation. Therefore, t is taken as the implicit control parameter. Accordingly, shape sensitivity is based on the idea of material derivative in continuum mechanics and proceeds as follows: If $\Omega \subset \mathbb{R}^d$ is the domain with sufficient smooth boundaries $\Gamma = \partial\Omega$, then Ω is a collection of material particles whose positions ($\vec{x}$) change with time, t. A smooth topological variation of Ω will lead to Ω_t so that the configuration of Ω_t is given by the new coordinates of the material particles in Ω_t. Ω_t is constructed in the form of the flow of a given velocity field, $\vec{v}(\vec{x},t)$, where $\vec{v}(\vec{x},t)$ is the material description of the velocity field characterizing the change in shape of the object. $\vec{v}(\vec{x},t)$ which is also the descent direction, is the most important quantity of interest in shape optimization as it provides the direction of flow of the boundaries as the object deforms towards the optimum shape. The descent direction is computed from the shape derivative of the cost functional.

2.1.1 Shape and Material Derivative of State Variables

For an arbitrary scalar variable, u, defined over Ω, the relationship between its material and shape derivatives along the direction, $\vec{v}$, is given as follows

$$\begin{aligned} \dot{u} = \frac{du}{dt} &= \frac{\partial u}{\partial t} + \nabla u \cdot \vec{v} \\ &= u' + \nabla u \cdot \vec{v} \end{aligned} \tag{2.1}$$

If $\vec{u}$ is a vector, the relationship is

$$\dot{\vec{u}} = \frac{\partial \vec{u}}{\partial t} + \nabla \vec{u}\, \vec{v} \tag{2.2}$$

$\dot{u}$: total derivative also known as material derivative of u. u': shape derivative of u

2.1.2 Shape Derivative of Cost Functional and Hadamard Formula

Perturbation of shape functional with respect to changes in object shape along $\vec{v}$ is called the shape derivative and is defined as

$$\dot{J} := \frac{d}{d\,t} J|_{t=0^+} = \lim_{t \to 0^+} \frac{J(\Omega_t) - J(\Omega)}{t} = \delta J(\Omega; \vec{v}) \tag{2.3}$$

Computation of shape derivative of the cost functional using Eqn. 2.3 can be very difficult. However, from Reynold's transport theorem, the equations described below can be used to compute the material derivative of functionals defined by integrals, in which the integrands and domains of integration depend on t.

For a cost functional, J, given as

$$J = \int_{\Omega} f(x_t)\, d\,\Omega \tag{2.4}$$

the shape derivative characterizing the deformation of the object as it moves from Ω to Ω_t in the direction $\vec{v}$ can be computed as follows, using the Reynold's transport equations

$$\begin{aligned} \delta J(\Omega; \vec{v}) &= \int_{\Omega} f' d\,\Omega + \int_{\partial\Omega} f \vec{v} \cdot \vec{n}\, d\,\Gamma \\ \delta J(\Omega; \vec{v}) &= \int_{\Omega} \dot{f} d\,\Omega + \int_{\Omega} f\, \nabla \cdot \vec{v}\, d\Omega \end{aligned} \tag{2.5}$$

$\dot{f}$ and f' are the material and shape derivatives of f respectively. For example, if f is independent of Ω_t, then

$$\delta J(\Omega;\vec{v}) = \int_{\partial\Omega} f\,\vec{v}\cdot\vec{n}\,d\,\Gamma \tag{2.6}$$

Eqn. 2.5 is also used in computing the material derivatives of the weak forms of the partial differential equations satisfied by the state and adjoint equations as will be seen in Chapter 3.

2.1.3 Hadamard Formula and Descent Direction

The Hadamard-Zolésio theorem (Zolésio 1992; Delfour and Zolésio 2001) states that under some regularity assumptions, the shape derivative equations of Eqn. 2.5, can be expressed as the scalar product of the normal component of the velocity field, $\vec{v}$, with some scalar shape gradient defined on the surface of the object to be optimized. The formula is given by

$$\delta J(\Omega;\vec{v}) = \int_{\partial\Omega} \nabla J\,v\,d\,\Gamma = \langle \nabla J, v\rangle_{\partial\Omega} \tag{2.7}$$

∇J is the shape gradient of the cost functional, $v = \vec{v}\cdot\vec{n}$, where $\vec{n}$ is the normal to the object boundaries. ∇J in general depends on the state and associated adjoint state variables. Considering that $\delta J(\Omega;\vec{v})$ is the variation of the cost functional with respect to shape changes, the updated cost functional on the updated object shape, Ω_t, can be written as

$$J(\Omega_t) = J(\Omega) + \delta J(\Omega;\vec{v}) + error \tag{2.8}$$

Cost functional is minimized by ensuring the $J(\Omega_t)$ is reduced during the optimization process. From Eqn. 2.8, this is achievable if $\delta J(\Omega;\vec{v}) < 0$. One way to achieve this is using a gradient based iterative algorithm which will require finding a descent direction, $\vec{v}$, to guarantee a reduction in the cost function. Similar to the procedure in classical optimization problems, the simplest choice of $\vec{v}$ to guarantee steepest descent of the cost function is $\vec{v} = -\nabla J\,\vec{n}$, since $\delta J(\Omega;\vec{V}) = -\int_{\partial\Omega} |\nabla J|^2\,d\,\Gamma < 0$. As a result, $J(\Omega_t) < J(\Omega)$. Thereafter, a new object domain is computed as

$$\Omega_t = \Omega + \alpha\,\vec{v} \tag{2.9}$$

Where α is a step size along the descent direction.

From the foregoing analyses, it is obvious that the primary task in shape optimization is deriving the scalar shape gradient for use in computing the descent direction along which the object is deformed to optimize its shape. In the next chapter, we apply these concepts to find the optimum shape for drag minimization problem.

Chapter 3
Application To Drag Minimization

3.1 Introduction

This project involves shape sensitivity analysis for flow around an arbitrarily shaped 2D object placed in the path of a viscous, incompressible fluid modeled by steady state Navier-Stokes equation. The objective is to determine the shape of the object that minimizes a cost functional, subject to fixed geometric constraint, the shape area. Control variable is the object shape while the cost functional is the fluid drag around the object. Since the fluid flow must satisfy Navier-Stokes equation, the state variables are fluid velocity, $\vec{u}$ and pressure, p. Fluid dynamic viscosity is μ. The 2D version of the problem is solved in this project and a hypothetical problem domain is sketched in Fig. 3.1. It consists of a bounded flow region, $\Omega \subset \mathbb{R}^2$, with lengths l_x and l_y. The boundaries of the flow region are defined by $\partial\Omega = \Gamma_{in} \cup \Gamma_{out} \cup \Gamma_+ \cup \Gamma_-$. We consider the case of fluid flowing from left to right so that Γ_{in} and Γ_{out} are inlet and outlet boundaries respectively. Γ_+ and Γ_- are top and bottom boundaries of the domain with no-slip boundary condition. Inside Ω, an object (Ω_b) with boundaries Γ_s is placed. It is the optimum shape of Ω_b which minimizes fluid drag that we seek to find. Ω_b must have an area of A_o. The admissible shapes of Ω_b must be continuous, bounded with lipschitz and non-self-intersecting boundaries.

3.1.1 Drag Functional

The project objective is: for all admissible shapes with area A_o, find the shape of Ω_b that minimizes the fluid drag . Fluid drag is the force exerted in flow direction, by a moving fluid on an obstacle placed along it's path. Mathematically, it is defined as

$$J(\Omega, \vec{u}, p) = -\vec{n}_\infty \int_{\Gamma_s} \sigma \cdot \vec{n} \tag{3.1}$$

$\vec{n}_\infty$, $\vec{n}$ are the unit vector in the flow direction and normal vector to obstacle boundary while σ is fluid stress. Since we consider flow in the x-direction, $\vec{n}_\infty = \vec{e}_x$

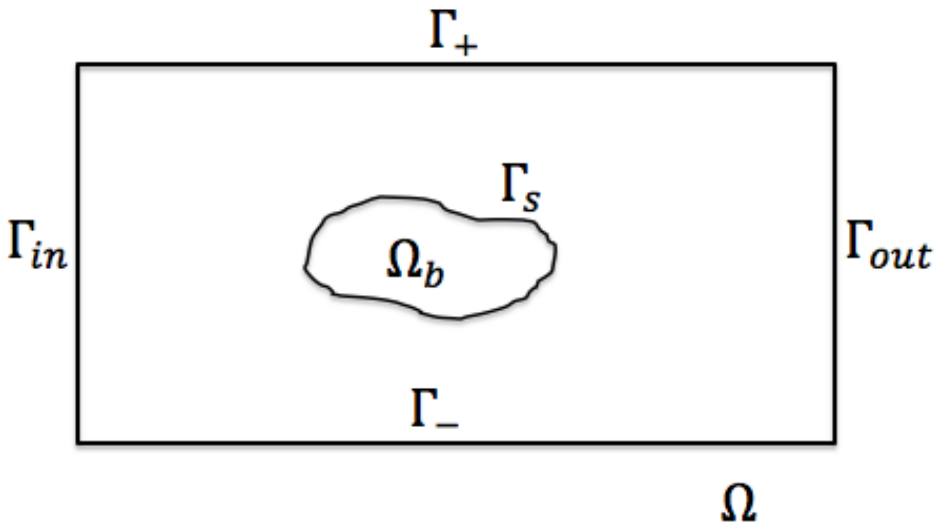

Figure 3.1: Conceptual domain for drag minimization problem

3.1.2 Constraints: Navier-Stokes Equation and Geometric Constraint

Two constraints are required to be satisfied by the fluid and the admissible shapes: a partial differential equation to model the state variables, velocity ($\vec{u}$) and pressure (p) and a specified area requirement for the admissible object.

Navier-Stokes and continuity equation is used to model fluid flow in the channel and around the obstacle. Both equations with applied boundary conditions are given as

$$-\nabla \cdot \sigma + (\vec{u} \cdot \nabla)\, \vec{u} = 0 \qquad \text{in } \Omega \tag{3.2}$$

$$\nabla \cdot \vec{u} = 0 \qquad \text{in } \Omega \tag{3.3}$$

$$\vec{u} = u_o \vec{e}_x \qquad \text{on } \Gamma_{in} \tag{3.4}$$

$$\vec{u} = 0 \qquad \text{on } \Gamma_+ \cup \Gamma_- \cup \Gamma_s \tag{3.5}$$

$$\sigma \cdot \vec{n} = 0 \qquad \text{in } \Gamma_{out} \tag{3.6}$$

Where

$$\sigma = -pI + 2\mu\,\varepsilon(\vec{u}) \tag{3.7}$$

$$\varepsilon(\vec{u}) = \frac{\nabla\vec{u} + \nabla\vec{u}^T}{2} \tag{3.8}$$

$$\tag{3.9}$$

$\varepsilon(\vec{u})$ is the fluid strain rate tensor.

In addition to PDE constraint for the fluid flow, an area constraint on the object is imposed so that area of the admissible shapes is fixed. Mathematical, the geometric constraint is given by

$$C := |\Omega_b| - A_o = 0 \tag{3.10}$$

3.2 Mathematical Analysis of Model Equations

3.2.1 Weak Formulation of Navier-Stokes Equation

Let $\vec{q} \in H_0^1(\Omega)$ and w be test functions for Navier-Stokes and continuity equations, the weak forms of Eqns. 3.2 and 3.3 are obtained as follows. Multiplying Eqns. 3.2 and 3.3 by $\vec{q}$ and w respectively, and integrating over Ω, we

$$\int_\Omega \big(-\nabla\cdot\sigma + (\vec{u}\cdot\nabla)\,\vec{u}\big)\cdot\vec{q} = 0 \tag{3.11}$$

From divergence theorem, we can write

$$\int_\Omega \nabla\cdot(\sigma\cdot\vec{q}) = \int_{\partial\Omega}(\sigma\cdot\vec{q})\cdot\vec{n} = \int_\Omega(\nabla\cdot\sigma)\cdot\vec{q} + \int_\Omega \sigma : \nabla\vec{q} \tag{3.12}$$

Therefore Eqn. 3.11 becomes

$$
\begin{aligned}
-\int_{\partial\Omega}(\sigma\cdot\vec{q})\cdot\vec{n}+\int_{\Omega}\sigma:\nabla\vec{q}+\int_{\Omega}(\vec{u}\cdot\nabla)\,\vec{u}\cdot\vec{q}=0 \\
\int_{\Omega}\sigma:\nabla\vec{q}+\int_{\Omega}(\vec{u}\cdot\nabla)\,\vec{u}\cdot\vec{q}=0 \quad (\sigma\cdot\vec{n}=0 \quad \text{from Eqn. 3.6}) \\
\int_{\Omega}\sigma:\nabla\vec{q}+\int_{\Omega}(\vec{u}\cdot\nabla)\,\vec{u}\cdot\vec{q}=0
\end{aligned}
\tag{3.13}
$$

If σ from Eqn. 3.89 is substituted into Eqn. 3.13, we have

$$
-\int_{\Omega}p\,\nabla\cdot\vec{q}+\int_{\Omega}2\mu\,\varepsilon(\vec{u}):\varepsilon(\vec{q})+\int_{\Omega}(\vec{u}\cdot\nabla)\,\vec{u}\cdot\vec{q}=0 \tag{3.14}
$$

The weak form of the continuity equation is simply

$$
\int_{\Omega}w\nabla\cdot\vec{u}=0 \tag{3.15}
$$

Adding Eqn. 3.14 and 3.15, the weak form of the Navier-Stokes equation becomes

$$
-\int_{\Omega}p\,\nabla\cdot\vec{q}+\int_{\Omega}2\mu\,\varepsilon(\vec{u}):\varepsilon(\vec{q})+\int_{\Omega}(\vec{u}\cdot\nabla)\,\vec{u}\cdot\vec{q}+\int_{\Omega}w\nabla\cdot\vec{u}=0 \tag{3.16}
$$

If we consider the following bilinear and trilinear forms,

$$
\begin{aligned}
a(\vec{u},\vec{q}) &= \int_{\Omega}2\mu\,\varepsilon(\vec{u}):\varepsilon(\vec{q}) \\
c(p,\vec{q}) &= \int_{\Omega}p\,\nabla\cdot\vec{q} \\
c(w,\vec{u}) &= \int_{\Omega}w\,\nabla\cdot\vec{u} \\
b(\vec{u},\vec{u},\vec{q}) &= \int_{\Omega}(\vec{u}\cdot\nabla)\,\vec{u}\cdot\vec{q}=\int_{\Omega}(\nabla\vec{u}\cdot\vec{u})\cdot\vec{q}
\end{aligned}
\tag{3.17}
$$

then an alternate form for the flow model is: find $\vec{u}$ and p such that

$$
a(\vec{u},\vec{q})+b(\vec{u},\vec{u},\vec{q})-c(p,\vec{q})-c(w,\vec{u})=0 \quad \forall \quad \vec{q}\text{ and }w \tag{3.18}
$$

3.2.2 Shape Derivative of State Variables, $\vec{u}$ and p

According to Reynold's transport equation of Eqn. 2.5, the material derivative of the integral of a function $F(t,x)$, is given by

$$\begin{aligned}\frac{d}{dt}\int_\Omega F(t,x)\,dx &= \int_\Omega \frac{\partial F(t,x)}{\partial t}dx + \int_{\partial\Omega_t} F(t,x)\,\vec{v}\cdot\vec{n}\,d\Gamma \\ &= \int_\Omega F'(t,x)dx + \int_{\partial\Omega_t} F(t,x)\,\vec{v}\cdot\vec{n}\,d\Gamma\end{aligned} \tag{3.19}$$

Applying Eqn. 3.19 to the weak form of the Navier-Stokes equations as written in Eqn. 3.18, we have

$$\begin{aligned}&a(\vec{u}',\vec{q}) + b(\vec{u}',\vec{u},\vec{q}) + b(\vec{u},\vec{u}',\vec{q}) - c(p',\vec{q}) - c(w,\vec{u}') \\ &\qquad + a(\vec{u},\vec{q}') + b(\vec{u},\vec{u},\vec{q}') - c(p,\vec{q}') - c(w',\vec{u}) \\ &\quad + \int_{\Gamma_s}\Big(- p\,\nabla\cdot\vec{q} + 2\mu\,\varepsilon(\vec{u}) : \varepsilon(\vec{q}) + (\vec{u}\cdot\nabla)\,\vec{u}\cdot\vec{q} + w\nabla\cdot\vec{u}\Big)\,\vec{v}\cdot\vec{n} = 0\end{aligned} \tag{3.20}$$

Since $\nabla\cdot\vec{u} = 0$ in Ω and $\vec{u} = 0$ on Γ_s, Eqn. 3.20 becomes

$$\begin{aligned}&a(\vec{u}',\vec{q}) + b(\vec{u}',\vec{u},\vec{q}) + b(\vec{u},\vec{u}',\vec{q}) - c(p',\vec{q}) - c(w,\vec{u}') \\ &\qquad + a(\vec{u},\vec{q}') + b(\vec{u},\vec{u},\vec{q}') - c(p,\vec{q}') - c(w',\vec{u}) + \int_{\Gamma_s}(\sigma : \nabla\vec{q})\,(\vec{v}\cdot\vec{n}) = 0\end{aligned} \tag{3.21}$$

The last row of the equation above is analyzed as follows: If we multiply Eqns. 3.2 and 3.3 by $\vec{q}'$ and w' and integrate over Ω, we obtain

$$a(\vec{u},\vec{q}') + b(\vec{u},\vec{u},\vec{q}') - c(p,\vec{q}') - c(w',\vec{u}) - \int_{\partial\Omega}\sigma\cdot\vec{n}\cdot\vec{q}' = 0 \tag{3.22}$$

The boundary term in Eqn. 3.22 is expanded as

$$\int_{\partial\Omega}\sigma\cdot\vec{n}\cdot\vec{q}' = \int_{\Gamma_{in}}\sigma\cdot\vec{n}\cdot\vec{q}' + \int_{\Gamma_+,\Gamma_-}\sigma\cdot\vec{n}\cdot\vec{q}' + \int_{\Gamma_{out}}\sigma\cdot\vec{n}\cdot\vec{q}' + \int_{\Gamma_s}\sigma\cdot\vec{n}\cdot\vec{q}' \tag{3.23}$$

Remember $\dot{\vec{q}} = \vec{q}' + \nabla\vec{q}\cdot\vec{v}$. Since $\vec{q} = 0$ on $\Gamma_{in} \cup \Gamma_+ \cup \Gamma_-$, then $\dot{\vec{q}} = 0$ on those boundaries and $\vec{q}' = -\nabla\vec{q}\cdot\vec{v}$. However, $\Gamma_{in} \cup \Gamma_+ \cup \Gamma_-$ do not deform. Therefore, $\vec{v} = 0$ on $\Gamma_{in} \cup \Gamma_+ \cup \Gamma_-$. In addition, $\sigma \cdot \vec{n} = 0$ on Γ_{out}. As a result, the only non-zero term of Eqn. 3.23 is on Γ_s. Eqn. 3.23 becomes

$$\int_{\partial\Omega} \sigma \cdot \vec{n} \cdot \vec{q}' = -\int_{\Gamma_s} \sigma \cdot \vec{n} \cdot (\nabla\vec{q} \cdot \vec{v}) \tag{3.24}$$

Applying the same analysis as in Eqn. A.10, A.11, A.12, A.13 and considering that $\nabla_\Gamma \vec{q} = 0$ since $\vec{q} = 0$ on Γ_s, we have

$$\int_{\partial\Omega} \sigma \cdot \vec{n} \cdot \vec{q}' = -\int_{\Gamma_s} (\sigma : \nabla\vec{q})\,(\vec{v} \cdot \vec{n}) \tag{3.25}$$

Substituting Eqn. 3.25 in Eqn. 3.22, we obtain

$$a(\vec{u}, \vec{q}') + b(\vec{u}, \vec{u}, \vec{q}') - c(p, \vec{q}') - c(w', \vec{u}) + \int_{\Gamma_s} (\sigma : \nabla\vec{q})\,(\vec{v} \cdot \vec{n}) = 0 \tag{3.26}$$

Upon substituting Eqn. 3.26 back into Eqn. 3.21, the weak form equation for shape derivative of the state variables is

$$a(\vec{u}', \vec{q}) + b(\vec{u}', \vec{u}, \vec{q}) + b(\vec{u}, \vec{u}', \vec{q}) - c(p', \vec{q}) - c(w, \vec{u}') = 0 \tag{3.27}$$

The boundary conditions are analyzed as follows.

$$\dot{\vec{u}} = \vec{u}' + \nabla\vec{u} \cdot \vec{v} \tag{3.28}$$

On all Dirichlet boundaries ($\Gamma_{in} \cup \Gamma_s \cup \Gamma_+ \cup \Gamma_-$), $\vec{u}$ is fixed. Therefore, $\dot{\vec{u}} = 0$ and

$$\vec{u}' = -\nabla\vec{u} \cdot \vec{v} \tag{3.29}$$

Since the coordinates of $\Gamma_{in} \cup \Gamma_+ \cup \Gamma_-$ do not change, $\vec{v} = 0$ on those boundaries and

$$\vec{u}' = 0 \qquad \text{on } \Gamma_{in} \cup \Gamma_+ \cup \Gamma_- \tag{3.30}$$

$$\vec{u}' = -\nabla \vec{u} \cdot \vec{v} \qquad \text{on } \Gamma_s \tag{3.31}$$

Therefore, the strong form of the equations satisfied by the shape derivatives of pressure and velocity, $\vec{u}'$ and p' are

$$-\nabla \cdot \sigma' + \vec{u}' \cdot \nabla \vec{u} + (\vec{u} \cdot \nabla)\, \vec{u}' = 0 \qquad \text{in } \Omega \tag{3.32}$$

$$\nabla \cdot \vec{u}' = 0 \qquad \text{in } \Omega \tag{3.33}$$

$$\vec{u}' = 0 \qquad \text{on } \Gamma_{in} \cup \Gamma_+ \cup \Gamma_- \tag{3.34}$$

$$\vec{u}' = -\nabla \vec{u} \cdot \vec{v} \qquad \text{on } \Gamma_s \tag{3.35}$$

$$\sigma' \cdot \vec{n} = 0 \qquad \text{on } \Gamma_{out} \tag{3.36}$$

Where

$$\sigma = -p' I + 2\mu\, \varepsilon(\vec{u}') \tag{3.37}$$

3.2.3 Cost Functional

Following the approach by Morin et al. (2011), Dede (2007) and Brandenburg et al. (2009), an equivalent volume integral of the cost function, Eqn. (3.1), is formulated as follows. Let ϕ be a function defined over the domain as below.

$$\vec{\phi} = \begin{cases} -\vec{n}_\infty & \text{on } \Gamma_s \\ 0 & \text{a.e} \end{cases}$$

Also, let

$$I(\Omega, \vec{u}, p) = \int_{\partial\Omega} \vec{\phi}\,\sigma \cdot \vec{n} = \int_{\Gamma_s} \vec{\phi}\,\sigma \cdot \vec{n} + \int_{\Gamma_{in}} \vec{\phi}\,\sigma \cdot \vec{n} + \int_{\Gamma_{out}} \vec{\phi}\,\sigma \cdot \vec{n} + \int_{\Gamma_+} \vec{\phi}\,\sigma \cdot \vec{n} + \int_{\Gamma_-} \vec{\phi}\,\sigma \cdot \vec{n} \tag{3.38}$$

However, since $\sigma \cdot \vec{n} = 0$ on Γ_{out} and $\vec{\phi} = 0$ on Γ_{in}, Γ_+ and Γ_-, Eqn. 3.38 becomes

$$J(\Omega, \vec{u}, p) = \int_{\Gamma_s} \vec{\phi}\,\sigma \cdot \vec{n} = J(\Omega, \vec{u}, p) \tag{3.39}$$

Therefore we can write that

$$J(\Omega, \vec{u}, p) = \int_{\partial\Omega} \vec{\phi}\,\sigma \cdot \vec{n} \tag{3.40}$$

Applying Green's theorem to Eqn. 3.40

$$\begin{aligned} J(\Omega, \vec{u}, p) &= \int_{\partial\Omega} \vec{\phi}\,\sigma \cdot \vec{n} = \int_{\Omega} \nabla \cdot (\sigma \cdot \vec{\phi}) \\ &= \int_{\Omega} \vec{\phi} \cdot \nabla \cdot \sigma + \int_{\Omega} \nabla\vec{\phi} : \sigma \\ &= \int_{\Omega} (\vec{u} \cdot \nabla)\,\vec{u} \cdot \vec{\phi} + \int_{\Omega} \nabla\vec{\phi} : \sigma \quad (\text{since } \nabla \cdot \sigma = \vec{u} \cdot \nabla \cdot \vec{u}) \\ &= \int_{\Omega} (\vec{u} \cdot \nabla)\,\vec{u} \cdot \vec{\phi} - \int_{\Omega} p\,\nabla \cdot \vec{\phi} + \int_{\Omega} 2\mu\,\varepsilon(\vec{\phi}) : \varepsilon(\vec{u}) \\ &= a(\vec{u}, \vec{\phi}) + b(\vec{u}, \vec{u}, \vec{\phi}) - c(p, \vec{\phi}) \end{aligned} \tag{3.41}$$

3.3 Unconstrained Optimization: Lagrangian Method

The optimization problem involves finding Ω_b among all admissible configurations with fixed area, that minimizes Eqn. 3.41 subject to $\vec{u}$ and p satisfying Navier-Stokes equations. The problem is formally converted to an unconstrained optimization problem by constructing a Lagrangian functional to remove the state and geometric constraints as shown below. For all $\vec{q}$ and w and for λ as Lagrange multiplier, the Lagrangian is

$$\begin{aligned}\mathcal{L}\Big(\Omega,(\vec{u},p),(\vec{q},w),\lambda\Big) &= J(\Omega,\vec{u},p) - a(\vec{u},\vec{q}) - b(\vec{u},\vec{u},\vec{q}) + c(p,\vec{q}) + c(w,\vec{u}) + \lambda\, C \\ &= a(\vec{u},\vec{\phi}) + b(\vec{u},\vec{u},\vec{\phi}) - c(p,\vec{\phi}) - a(\vec{u},\vec{q}) - b(\vec{u},\vec{u},\vec{q}) + c(p,\vec{q}) + c(w,\vec{u}) + \lambda\, C \\ &= a(\vec{u},\vec{\phi}-\vec{q}) + b(\vec{u},\vec{u},\vec{\phi}-\vec{q}) - c(p,\vec{\phi}-\vec{q}) + c(w,\vec{u}) + \lambda\, C\end{aligned} \tag{3.42}$$

If $\vec{z} = \vec{\phi} - \vec{q}$ and $r = -w$, Eqn. 3.42 becomes

$$\mathcal{L}\Big(\Omega,(\vec{u},p),(\vec{z},r),\lambda\Big) = a(\vec{u},\vec{z}) + b(\vec{u},\vec{u},\vec{z}) - c(p,\vec{z}) - c(r,\vec{u}) + \lambda\, C \tag{3.43}$$

where $\vec{z}$ and r are the adjoint variables. Applying the KKT conditions, the derivative of $\mathcal{L}\Big(\Omega,(\vec{u},p),(\vec{z},r),\lambda\Big)$ with respect to Ω, $(\vec{u},p)$, $(\vec{z},r)$ and λ have to be zero at the optimum solution. That is

$$\delta\mathcal{L}\Big(\Omega,(\vec{u},p),(\vec{z},r),\lambda;\vec{v}\Big) = 0 \tag{3.44}$$

$$\delta_{\vec{u},p}\mathcal{L}\Big(\Omega,(\vec{u},p),(\vec{z},r),\lambda;(\vec{v}_o,h_o)\Big) = 0 \tag{3.45}$$

$$\delta_{\vec{z},r}\mathcal{L}\Big(\Omega,(\vec{u},p),(\vec{z},r),\lambda;(\vec{v}_1,h_1)\Big) = 0 \tag{3.46}$$

$$\nabla_\lambda\mathcal{L}\Big(\Omega,(\vec{u},p),(\vec{z},r),\lambda\Big) = 0 \tag{3.47}$$

Eqn. 3.46 is the derivative along the arbitrary $(\vec{v}_o,h_o)$ direction, of the Lagrangian function with respect to the adjoint variables. It simply leads to the Navier-Stokes equation which will be solved as part of the solution of the optimization process. Eqn. 3.45 is derivative of the Lagrangian function with respect to the state variables, along an arbitrary $(\vec{v}_o,h_o)$ direction. It generates the partial differential equations that the adjoint variables must satisfy. Eqn. 3.44 is the material derivative of the Lagrangian function and from it's Hadamard form (Eqn. 2.7), the shape gradient will be obtained and used to move the boundary nodes of the obstacle. As observed in Eqn. 3.43, the Lagrangian function depends on the state and

adjoint variables. Therefore, solutions of Eqn. 3.45 and 3.46 will be required to obtain the correct value for the shape gradient. Derivative of the Lagrangian with respect to the Lagrange multiplier is simply

$$\nabla_\lambda \mathcal{L}\Big(\Omega, (\vec{u}, p), (\vec{z}, r), \lambda\Big) = C \tag{3.48}$$

It is important to note that Eqn. 3.48 is not the optimality condition. Rather, it is the value of the derivative at any point during the optimization process. However, once the geometric constraint is satisfied at the end of optimization (i.e. $C = 0$), then the optimality condition $\nabla_\lambda \mathcal{L}\Big(\Omega, (\vec{u}, p), (\vec{z}, r), \lambda\Big) = 0$ is also satisfied.

3.3.1 Derivation of Adjoint Equation

Derivative of Eqn. 3.42 with respect to the state ($\vec{u}$ and p) along $\vec{v}$ and h direction proceeds as follows:

$$\begin{aligned}
\delta_{\vec{u},p}\mathcal{L}\Big(\Omega, (\vec{u}, p), (\vec{z}, r), \lambda; (\vec{v}_o, h_o)\Big) &= a(\vec{v}_o, \vec{z}) + b(\vec{v}_o, \vec{u}, \vec{z}) + b(\vec{u}, \vec{v}_o, \vec{z}) - c(h_o, \vec{z}) - c(r, \vec{v}) \\
&= \int_\Omega 2\mu\, \varepsilon(\vec{v}_o) : \varepsilon(\vec{z}) + \int_\Omega (\vec{v}_o \cdot \nabla)\, \vec{u} \cdot \vec{z} + \int_\Omega (\vec{u} \cdot \nabla)\, \vec{v}_o \cdot \vec{z} \\
&\quad - \int_\Omega h_o\, \nabla \cdot \vec{z} - \int_\Omega r\, \nabla \cdot \vec{v}_o \\
&= \int_\Omega 2\mu\, \varepsilon(\vec{v}_o) : \varepsilon(\vec{z}) - \int_\Omega r\, \nabla \cdot \vec{v}_o + \int_\Omega (\vec{v}_o \cdot \nabla)\, \vec{u} \cdot \vec{z} \\
&\quad + \int_\Omega (\vec{u} \cdot \nabla)\, \vec{v}_o \cdot \vec{z} - \int_\Omega h_o\, \nabla \cdot \vec{z} \\
&= 0
\end{aligned} \tag{3.49}$$

If we define T as

$$T = -rI + 2\mu\, \varepsilon(\vec{z}) \tag{3.50}$$

then,

$$\begin{aligned}\delta_{\vec{u},p}\mathcal{L}\Big(\Omega,(\vec{u},p),(\vec{z},r),\lambda;(\vec{v}_o,h_o)\Big) &= \int_\Omega T:\nabla\vec{v}_o+\int_\Omega(\vec{v}_o\cdot\nabla)\,\vec{u}\cdot\vec{z}+\int_\Omega(\vec{u}\cdot\nabla)\,\vec{v}_o\cdot\vec{z}-\int_\Omega h_o\,\nabla\cdot\vec{z}\\ &= a(\vec{z},\vec{v}_o)-c(r,\vec{v}_o)+b(\vec{u},\vec{v}_o,\vec{z})+b(\vec{v}_o,\vec{u},\vec{z})-c(h_o,\vec{z})\\ &= 0\end{aligned} \tag{3.51}$$

Eqn. 3.51 above is the weak form equation satisfied by the adjoint variables. To derive the strong form of Eqn. 3.51, we carry out integration by parts of the equation to return it to its pristine form after it has just been multiplied by the test functions $\vec{v}_o$ and h_o. The trilinear form are further analyzed as follows. (see Lemma 6.3 in Slawig (2003) and Slawig (2006) for more information).

$$\int_\Omega\nabla\cdot\big((\vec{v}_o\cdot\vec{z})\,\vec{u}\big)=\int_{\partial\Omega}(\vec{v}_o\cdot\vec{z})\,(\vec{u}\cdot\vec{n}) \qquad =\int_\Omega(\vec{v}_o\cdot\vec{z})\,\nabla\cdot\vec{u}+\int_\Omega\vec{u}\cdot\nabla(\vec{v}_o\cdot\vec{z}) \tag{3.52}$$

$$=\int_\Omega\vec{u}\cdot\nabla(\vec{v}_o\cdot\vec{z})\quad(\text{ since }\nabla\cdot\vec{u}=0) \tag{3.53}$$

The right hand side is further analyzed using indicial notations

$$\begin{aligned}\int_\Omega\vec{u}\cdot\nabla(\vec{v}_o\cdot\vec{z}) &= \int_\Omega u_j\frac{\partial\,(v_{o_i}\,z_i)}{\partial x_j}\\ &= \int_\Omega u_j v_{o_i}\frac{\partial\,z_i}{\partial x_j}+\int_\Omega u_j z_i\frac{\partial\,v_{o_i}}{\partial x_j}\\ &= \int_\Omega u_j\frac{\partial\,z_i}{\partial x_j}v_{o_i}+\int_\Omega u_j\frac{\partial\,v_{o_i}}{\partial x_j}z_i\\ &= \int_\Omega(\vec{u}\cdot\nabla)\,\vec{z}\cdot\vec{v}_o+\int_\Omega(\vec{u}\cdot\nabla)\,\vec{v}_o\cdot\vec{z}\end{aligned} \tag{3.54}$$

Substituting Eqn. 3.54 into Eqn. 3.53, we have

$$\int_{\partial\Omega}(\vec{v}_o\cdot\vec{z})\,(\vec{u}\cdot\vec{n})=\int_\Omega(\vec{u}\cdot\nabla)\,\vec{z}\cdot\vec{v}_o+\int_\Omega(\vec{u}\cdot\nabla)\,\vec{v}_o\cdot\vec{z} \tag{3.55}$$

Therefore,

$$\int_{\Omega} (\vec{u} \cdot \nabla)\, \vec{v}_o \cdot \vec{z} = \int_{\partial\Omega} (\vec{v}_o \cdot \vec{z})\, (\vec{u} \cdot \vec{n}) - \int_{\Omega} (\vec{u} \cdot \nabla)\, \vec{z} \cdot \vec{v}_o \tag{3.56}$$

After considering divergence formula for the first term on the right hand side of Eqn. 3.51 and upon substituting Eqn. 3.56 into Eqn. 3.51, we have

$$\begin{aligned} \delta_{\vec{u},p}\mathcal{L}\Big(\Omega, (\vec{u}, p), (\vec{z}, r), \lambda; (\vec{v}_o, h_o)\Big) = & \int_{\partial\Omega} T \cdot \vec{v}_o \cdot \vec{n} - \int_{\Omega} \nabla \cdot T \cdot \vec{v}_o + \int_{\Omega} (\nabla \vec{u}^T \cdot \vec{z}) \cdot \vec{v}_o \\ & - \int_{\Omega} (\vec{u} \cdot \nabla)\, \vec{z} \cdot \vec{v}_o + \int_{\partial\Omega} (\vec{u} \cdot \vec{n})(\vec{z} \cdot \vec{v}_o) - \int_{\Omega} h_o \nabla \cdot \vec{z} \\ = & \, 0 \end{aligned} \tag{3.57}$$

Remember that from Eqn. 3.11, $\vec{q}$ is a test function so that it has a value of zero on the Dirichlet boundaries of the Navier-Stokes problem i.e

$$\vec{q} = 0 \text{ on } \Gamma_s \cup \Gamma_{in} \cup \Gamma_+ \cup \Gamma_- \tag{3.58}$$

Therefore,

$$\vec{z} = \vec{\phi} - \vec{q} = \vec{\phi} \text{ on } \Gamma_s \cup \Gamma_{in} \cup \Gamma_+ \cup \Gamma_- \tag{3.59}$$

Eqn. 3.59 means that Γ_s, Γ_{in}, Γ_+ and Γ_- are also Dirichlet boundaries for the adjoint problem while Γ_{out} is a Neumann boundary. The strong form of Eqn. 3.49 therefore becomes

$$-\nabla \cdot T + \nabla \vec{u}^T \cdot \vec{z} - (\vec{u} \cdot \nabla)\, \vec{z} = 0 \qquad \text{in } \Omega \tag{3.60}$$

$$\nabla \cdot \vec{z} = 0 \qquad \text{in } \Omega \tag{3.61}$$

$$\vec{z} = \vec{\phi} \qquad \text{on } \Gamma_{in} \cup \Gamma_+ \cup \Gamma_- \cup \Gamma_s \tag{3.62}$$

$$T \cdot \vec{n} + (\vec{u} \cdot \vec{n})\, \vec{z} = 0 \qquad \text{in } \Gamma_{out} \tag{3.63}$$

where

$$T = -rI + 2\mu\,\varepsilon(\vec{z}) \tag{3.64}$$

3.3.2 Shape Derivative of Adjoint Variables, $\vec{z}$ and r

The weak form of the adjoint equation is simply Eqn. 3.51, shown below for convenience. $\vec{v}_o$ and h_o are test functions.

$$a(\vec{z},\vec{v}_o) - c(r,\vec{v}_o) + b(\vec{u},\vec{v}_o,\vec{z}) + b(\vec{v}_o,\vec{u},\vec{z}) - c(h_o,\vec{z}) = 0 \tag{3.65}$$

Upon applying Eqn. 2.5 to the equation above, we have

$$\begin{aligned} &a(\vec{z}',\vec{v}_o) - c(r',\vec{v}_o) + b(\vec{u}',\vec{v}_o,\vec{z}) + b(\vec{u},\vec{v}_o,\vec{z}') + b(\vec{v}_o,\vec{u}',\vec{z}) + b(\vec{v}_o,\vec{u},\vec{z}') - c(h_o,\vec{z}') \\ &\quad + a(\vec{z},\vec{v}_o') - c(r,\vec{v}_o') + b(\vec{u},\vec{v}_o',\vec{z}) + b(\vec{v}_o',\vec{u},\vec{z}) - c(h_o',\vec{z}) \\ &\quad + \int_{\Gamma_s} \Big(T : \nabla\vec{v}_o + (\nabla\vec{u}^T\cdot\vec{z})\cdot\vec{v}_o + (\vec{u}\cdot\nabla)\,\vec{v}_o\cdot\vec{z} - h_o\,\nabla\cdot\vec{z}\Big)\vec{v}\cdot\vec{n} = 0 \end{aligned} \tag{3.66}$$

Again, since $\vec{u} = 0$ on Γ_s, $\vec{v}_o = 0$ on Γ_s and $\nabla\cdot\vec{z} = 0$ in Ω we can write the above equation as

$$\begin{aligned} &a(\vec{z}',\vec{v}_o) - c(r',\vec{v}_o) + b(\vec{u}',\vec{v}_o,\vec{z}) + b(\vec{u},\vec{v}_o,\vec{z}') + b(\vec{v}_o,\vec{u}',\vec{z}) + b(\vec{v}_o,\vec{u},\vec{z}') - c(h_o,\vec{z}') \\ &\quad + a(\vec{z},\vec{v}_o') - c(r,\vec{v}_o') + b(\vec{u},\vec{v}_o',\vec{z}) + b(\vec{v}_o',\vec{u},\vec{z}) - c(h_o',\vec{z}) + \int_{\Gamma_s} (T : \nabla\vec{v}_o)\vec{v}\cdot\vec{n} = 0 \end{aligned} \tag{3.67}$$

Insight into further analysis of the last line of equation Eqn. 3.67 is obtained by multiplying Eqns. 3.60 and 3.61 by $\vec{v}_o'$ and h_o' and integrating over Ω.

$$\begin{aligned} -\int_\Omega \nabla\cdot T\cdot\vec{v}_o' + \int_\Omega \nabla\vec{u}^T\cdot\vec{z}\cdot\vec{v}_o' - \int_\Omega (\vec{u}\cdot\nabla)\,\vec{z}\cdot\vec{v}_o' - \int_\Omega h_o'\,\nabla\cdot\vec{z} = 0 \\ \int_\Omega T : \nabla\vec{v}_o' - \int_{\partial\Omega} T\cdot\vec{n}\cdot\vec{v}_o' + \int_\Omega (\vec{v}_o'\cdot\nabla)\,\vec{u}\cdot\vec{z} - \int_\Omega (\vec{u}\cdot\nabla)\,\vec{z}\cdot\vec{v}_o' - \int_\Omega h_o'\,\nabla\cdot\vec{z} = 0 \end{aligned} \tag{3.68}$$

If we make use of Eqn. 3.56, we have

$$\int_\Omega T : \nabla \vec{v}_o' + \int_\Omega (\vec{v}_o' \cdot \nabla)\, \vec{u} \cdot \vec{z} + \int_\Omega (\vec{u} \cdot \nabla)\, \vec{v}_o \cdot \vec{z}' - \int_\Omega h_o' \nabla \cdot \vec{z} - \int_{\partial\Omega} \left(T \cdot \vec{n} + (\vec{u} \cdot \vec{n})\, \vec{z} \right) \cdot \vec{v}_o' = 0$$
$$a(\vec{z}, \vec{v}_o') - c(r, \vec{v}_o') + b(\vec{u}, \vec{v}_o', \vec{z}) + b(\vec{v}_o', \vec{u}, \vec{z}) - c(h_o', \vec{z}) - \int_{\partial\Omega} \left(T \cdot \vec{n} + (\vec{u} \cdot \vec{n})\, \vec{z} \right) \cdot \vec{v}_o' = 0 \tag{3.69}$$

But

$$\int_{\partial\Omega} \left(T \cdot \vec{n} + (\vec{u} \cdot \vec{n})\, \vec{z} \right) \cdot \vec{v}_o' = \int_{\Gamma_{in}} \left(T \cdot \vec{n} + (\vec{u} \cdot \vec{n})\, \vec{z} \right) \cdot \vec{v}_o' + \int_{\Gamma_+, \Gamma_-} \left(T \cdot \vec{n} + (\vec{u} \cdot \vec{n})\, \vec{z} \right) \cdot \vec{v}_o' + \int_{\Gamma_s} \left(T \cdot \vec{n} + (\vec{u} \cdot \vec{n})\, \vec{z} \right) \cdot \vec{v}_o'$$
$$+ \int_{\Gamma_{out}} \left(T \cdot \vec{n} + (\vec{u} \cdot \vec{n})\, \vec{z} \right) \cdot \vec{v}_o' \tag{3.70}$$

$T \cdot \vec{n} + (\vec{u} \cdot \vec{n})\, \vec{z} = 0$ on Γ_{out}. In addition, total derivative (material derivative) for $\vec{v}_o$ is

$$\dot{\vec{v}}_o = \vec{v}_o' + \nabla \vec{v}_o \cdot \vec{v} \tag{3.71}$$

Since $\vec{v}_o$ is fixed on $\Gamma_s \cup \Gamma_{in} \cup \Gamma_+ \cup \Gamma_-$, this means that $\dot{\vec{v}}_o = 0$ and $\vec{v}_o' = -\nabla \vec{v}_o \cdot \vec{v}$.
But on $\Gamma_{in} \cup \Gamma_+ \cup \Gamma_-$, $\vec{v} = \vec{0}$ since these boundaries do not move. Therefore, the boundary conditions for $\vec{v}_o'$ on $\Gamma_s \cup \Gamma_{in} \cup \Gamma_+ \cup \Gamma_-$ are

$$\vec{v}_o' = 0 \qquad \text{on } \Gamma_{in} \cup \Gamma_+ \cup \Gamma_- \tag{3.72}$$
$$\vec{v}_o' = -\nabla \vec{v}_o \cdot \vec{v} \qquad \text{on } \Gamma_s \tag{3.73}$$

Therefore, Eqn. 3.74 is

$$\int_{\partial\Omega} \left(T \cdot \vec{n} + (\vec{u} \cdot \vec{n})\, \vec{z} \right) \cdot \vec{v}_o' = \int_{\Gamma_s} \left(T \cdot \vec{n} + (\vec{u} \cdot \vec{n})\, \vec{z} \right) \cdot \vec{v}_o' = -\int_{\Gamma_s} \left(T \cdot \vec{n} + (\vec{u} \cdot \vec{n})\, \vec{z} \right) \cdot (\nabla \vec{v}_o \cdot \vec{v})$$
$$= -\int_{\Gamma_s} T \cdot \vec{n} \cdot (\nabla \vec{v}_o \cdot \vec{v}) \quad (\vec{u} = 0 \text{ on } \Gamma_s) \tag{3.74}$$

Applying the same analysis as in Eqn. A.10, A.11, A.12, A.13 and considering that $\nabla_\Gamma \vec{v}_o = 0$ since $\vec{v}_o = 0$ on Γ_s, we have

$$\int_{\partial\Omega} \left(T \cdot \vec{n} + (\vec{u} \cdot \vec{n})\, \vec{z} \right) \cdot \vec{v}_o' = - \int_{\Gamma_s} (T : \nabla \vec{v}_o)\, (\vec{v} \cdot \vec{n}) \tag{3.75}$$

Substituting Eqn. 3.75 into Eqn. 3.69, we

$$a(\vec{z}, \vec{v}_o') - c(r, \vec{v}_o') + b(\vec{u}, \vec{v}_o', \vec{z}) + b(\vec{v}_o', \vec{u}, \vec{z}) - c(h_o', \vec{z}) + \int_{\Gamma_s} (T : \nabla \vec{v}_o)\, (\vec{v} \cdot \vec{n}) = 0 \tag{3.76}$$

Therefore, substituting Eqn. 3.76 into Eqn. 3.67, the weak form equation satisfied by $\vec{z}'$ and r' is

$$a(\vec{z}', \vec{v}_o) - c(r', \vec{v}_o) + b(\vec{u}', \vec{v}_o, \vec{z}) + b(\vec{u}, \vec{v}_o, \vec{z}') + b(\vec{v}_o, \vec{u}', \vec{z}) + b(\vec{v}_o, \vec{u}, \vec{z}') - c(h_o, \vec{z}') = 0 \tag{3.77}$$

The expanded form of the above equation is

$$\begin{aligned} 0 = & \int_\Omega T' : \nabla \vec{v}_o + \int_\Omega (\nabla \vec{u}'^T \cdot \vec{z}) \cdot \vec{v}_o + \int_\Omega (\nabla \vec{u}^T \cdot \vec{z}') \cdot \vec{v}_o + \int_\Omega (\vec{u}' \cdot \nabla)\, \vec{v}_o \cdot \vec{z} + \int_\Omega (\vec{u} \cdot \nabla)\, \vec{v}_o \cdot \vec{z}' \\ & - \int_\Omega h_o \nabla \cdot \vec{z}' \\ = & \int_{\partial\Omega} T' \cdot \vec{v}_o \cdot \vec{n} - \int_\Omega \nabla \cdot T' \cdot \vec{v}_o + \int_\Omega (\nabla \vec{u}'^T \cdot \vec{z}) \cdot \vec{v}_o + \int_\Omega (\nabla \vec{u}^T \cdot \vec{z}') \cdot \vec{v}_o + \int_\Omega (\vec{u}' \cdot \nabla)\, \vec{v}_o \cdot \vec{z} \\ & + \int_\Omega (\vec{u} \cdot \nabla)\, \vec{v}_o \cdot \vec{z}' - \int_\Omega h_o \nabla \cdot \vec{z}' \end{aligned} \tag{3.78}$$

If we carrying out similar analysis as in Eqns. 3.53, 3.54, 3.55 and 3.56, we can express the 4th and 5th terms on the right hand side of Eqn. 3.78 as

$$\begin{aligned} \int_\Omega (\vec{u}' \cdot \nabla)\, \vec{v}_o \cdot \vec{z} &= - \int_\Omega (\vec{u}' \cdot \nabla)\, \vec{z} \cdot \vec{v}_o + \int_{\partial\Omega} (\vec{u}' \cdot \vec{n})(\vec{z} \cdot \vec{v}_o) \\ \int_\Omega (\vec{u} \cdot \nabla)\, \vec{v}_o \cdot \vec{z}' &= - \int_\Omega (\vec{u} \cdot \nabla)\, \vec{z}' \cdot \vec{v}_o + \int_{\partial\Omega} (\vec{u} \cdot \vec{n})(\vec{z}' \cdot \vec{v}_o) \end{aligned} \tag{3.79}$$

Therefore, Eqn. 3.78 becomes

$$0 = \int_{\partial\Omega} T' \cdot \vec{v}_o \cdot \vec{n} - \int_\Omega \nabla \cdot T' \cdot \vec{v}_o + \int_\Omega (\nabla \vec{u}'^T \cdot \vec{z}) \cdot \vec{v}_o + \int_\Omega (\nabla \vec{u}^T \cdot \vec{z}') \cdot \vec{v}_o - \int_\Omega (\vec{u}' \cdot \nabla)\, \vec{z} \cdot \vec{v}_o + \int_{\partial\Omega} (\vec{u}' \cdot \vec{n})(\vec{z} \cdot \vec{v}_o) - \int_\Omega (\vec{u} \cdot \nabla)\, \vec{z}' \cdot \vec{v}_o + \int_{\partial\Omega} (\vec{u} \cdot \vec{n})(\vec{z}' \cdot \vec{v}_o) - \int_\Omega h_o \nabla \cdot \vec{z}' \tag{3.80}$$

The total derivative (material derivative) for $\vec{z}$ is

$$\dot{\vec{z}} = \vec{z}' + \nabla \vec{z} \cdot \vec{v} \tag{3.81}$$

since $\vec{z}$ is fixed on $\Gamma_s \cup \Gamma_{in} \cup \Gamma_+ \cup \Gamma_-$, this means that $\dot{\vec{z}} = 0$ and $\vec{z}' = -\nabla \vec{z} \cdot \vec{v}$.
But on $\Gamma_{in} \cup \Gamma_+ \cup \Gamma_-$, $\vec{v} = \vec{0}$ since these boundaries do not move. Therefore, the boundary conditions of $\vec{z}'$ on $\Gamma_s \cup \Gamma_{in} \cup \Gamma_+ \cup \Gamma_-$ are

$$\vec{z}' = 0 \qquad \text{on } \Gamma_{in} \cup \Gamma_+ \cup \Gamma_- \tag{3.82}$$

$$\vec{z}' = -\nabla \vec{z} \cdot \vec{v} \qquad \text{on } \Gamma_s \tag{3.83}$$

Therefore, the strong form of the PDE satisfied by $\vec{z}'$ and r' are

$$-\nabla \cdot T' + (\nabla \vec{u}'^T \cdot \vec{z}) + (\nabla \vec{u}^T \cdot \vec{z}') - (\vec{u}' \cdot \nabla \vec{z}) - (\vec{u} \cdot \nabla \vec{z}') = 0 \qquad \text{in } \Omega \tag{3.84}$$

$$\nabla \cdot \vec{z}' = 0 \qquad \text{in } \Omega \tag{3.85}$$

$$\vec{z}' = 0 \qquad \text{on } \Gamma_{in} \cup \Gamma_+ \cup \Gamma_- \tag{3.86}$$

$$\vec{z}' = -\nabla \vec{u} \cdot \vec{v} \qquad \text{on } \Gamma_s \tag{3.87}$$

$$T' \cdot \vec{n} + (\vec{u}' \cdot \vec{n})\, \vec{z} + (\vec{u} \cdot \vec{n})\, \vec{z}' = 0 \qquad \text{on } \Gamma_{out} \tag{3.88}$$

Where

$$T' = -r'I + 2\mu\,\varepsilon(\vec{z}') \tag{3.89}$$

Note that Eqn. 3.88 is obtained as it is since $\vec{v} = 0$ on Γ_{out}. This means that, $\vec{u}' = 0$ and $\vec{z}' = 0$ on Γ_{out}.

3.3.3 Shape Sensitivity

The Lagrangian function (Eqn. 3.43) for the unconstrained problem is written below for convenience.

$$\mathcal{L}\Big(\Omega, (\vec{u}, p), (\vec{z}, r), \lambda\Big) = a(\vec{u}, \vec{z}) + b(\vec{u}, \vec{u}, \vec{z}) - c(p, \vec{z}) - c(r, \vec{u}) + \lambda\, C \tag{3.90}$$

Using Eqn. 3.19, the material derivative of the Lagrangian is

$$\begin{aligned}\delta\mathcal{L}\Big(\Omega, (\vec{u}, p), (\vec{z}, r), \lambda; \vec{v}\Big) = {} & a(\vec{u}', \vec{z}) + a(\vec{u}, \vec{z}') + b(\vec{u}', \vec{u}, \vec{z}) + b(\vec{u}, \vec{u}', \vec{z}) + b(\vec{u}, \vec{u}, \vec{z}') \\ & - c(p', \vec{z}) - c(r', \vec{u}) - c(p, \vec{z}') - c(r, \vec{u}') - \int_{\Gamma_s} p\nabla\cdot\vec{z}\,\vec{v}\cdot\vec{n} \\ & - \int_{\Gamma_s} r\nabla\cdot\vec{u}\,\vec{v}\cdot\vec{n} + 2\mu\int_{\Gamma_s}\varepsilon(\vec{z}) : \varepsilon(\vec{u})\vec{v}\cdot\vec{n} + \int_{\Gamma_s}(\vec{u}\cdot\nabla\vec{u})\cdot\vec{z}\,\vec{v}\cdot\vec{n} + \lambda\int_{\Gamma_s}\vec{v}\cdot\vec{n}\end{aligned} \tag{3.91}$$

Remember that $\nabla\cdot\vec{u} = 0$ and $\nabla\cdot\vec{z} = 0$ from Navier-Stokes and adjoint equation. Similarly, $\nabla\cdot\vec{u}' = 0$ and $\nabla\cdot\vec{z}' = 0$ from material derivative of state and adjoint variables. Therefore,

$$\begin{aligned}\delta\mathcal{L}\Big(\Omega, (\vec{u}, p), (\vec{z}, r), \lambda; \vec{v}\Big) = {} & a(\vec{u}', \vec{z}) + a(\vec{u}, \vec{z}') + b(\vec{u}', \vec{u}, \vec{z}) + b(\vec{u}, \vec{u}', \vec{z}) + b(\vec{u}, \vec{u}, \vec{z}') \\ & + 2\mu\int_{\Gamma_s}\varepsilon(\vec{z}) : \varepsilon(\vec{u})\vec{v}\cdot\vec{n} + \int_{\Gamma_s}(\vec{u}\cdot\nabla\vec{u})\cdot\vec{z}\,\vec{v}\cdot\vec{n} + \lambda\int_{\Gamma_s}\vec{v}\cdot\vec{n}\end{aligned} \tag{3.92}$$

Substituting Eqns. A.3 and A.6 into Eqn. 3.92 and considering that $(\vec{u}'\cdot\nabla)\,\vec{u}\cdot\vec{z} = (\nabla\vec{u}^T\cdot\vec{z})\cdot\vec{u}'$, Eqn. 3.92 becomes

$$\begin{aligned}\delta\mathcal{L}\Big(\Omega,(\vec{u},p),(\vec{z},r),\lambda;\vec{v}\Big) &= \int_{\Gamma_s} T\cdot\vec{n}\cdot\vec{u}' - \int_\Omega (\vec{u}'\cdot\nabla)\,\vec{u}\cdot\vec{z} + \int_\Omega (\vec{u}\cdot\nabla\vec{z})\cdot\vec{u}' + \int_{\Gamma_s}\sigma\cdot\vec{n}\cdot\vec{z}' \\ &\quad - \int_\Omega (\vec{u}\cdot\nabla)\,\vec{u}\cdot\vec{z}' + \int_\Omega (\vec{u}'\cdot\nabla)\,\vec{u}\cdot\vec{z} + \int_\Omega (\vec{u}\cdot\nabla)\,\vec{u}'\cdot\vec{z} + \int_\Omega (\vec{u}\cdot\nabla)\,\vec{u}\cdot\vec{z}' \\ &\quad + 2\mu\int_{\Gamma_s}\varepsilon(\vec{z}):\varepsilon(\vec{u})\vec{v}\cdot\vec{n} + \int_{\Gamma_s}(\vec{u}\cdot\nabla)\,\vec{u}\cdot\vec{z}\,\vec{v}\cdot\vec{n} + \lambda\int_{\Gamma_s}\vec{v}\cdot\vec{n} \\ &= \int_{\Gamma_s} T\cdot\vec{n}\cdot\vec{u}' + \int_{\Gamma_s}\sigma\cdot\vec{n}\cdot\vec{z}' + \int_\Omega (\vec{u}\cdot\nabla)\,\vec{z}\cdot\vec{u}' + \int_\Omega (\vec{u}\cdot\nabla)\,\vec{u}'\cdot\vec{z} \\ &\quad + 2\mu\int_{\Gamma_s}\varepsilon(\vec{z}):\varepsilon(\vec{u})\,\vec{v}\cdot\vec{n} + \int_{\Gamma_s}(\vec{u}\cdot\nabla)\,\vec{u}\cdot\vec{z}\,\vec{v}\cdot\vec{n} + \lambda\int_{\Gamma_s}\vec{v}\cdot\vec{n}\end{aligned} \tag{3.93}$$

Substituting Eqns. A.9, A.13 and A.14 into 3.93 considering that $\vec{u} = 0$ on Γ_s, we have

$$\begin{aligned}\delta\mathcal{L}\Big(\Omega,(\vec{u},p),(\vec{z},r),\lambda;\vec{v}\Big) &= -2\mu\int_{\Gamma_s}\varepsilon(\vec{z}):\varepsilon(\vec{u})\,v + \lambda\int_{\Gamma_s} v \\ &= \int_{\Gamma_s}\Big(-2\mu\,\varepsilon(\vec{z}):\varepsilon(\vec{u}) + \lambda\Big)\,\vec{v}\cdot\vec{n}\end{aligned} \tag{3.94}$$

Comparing Eqn. 3.94 with Hadamard's representation of shape derivative (Eqn. 2.7), the shape gradient for this drag minimization problem is

$$\nabla\mathcal{L} = -2\mu\,\varepsilon(\vec{z}):\varepsilon(\vec{u}) + \lambda \tag{3.95}$$

Thus, in order to minimize the fluid drag, Γ_s is deformed along the descent direction defined by $\vec{v} = -\nabla\mathcal{L}\,\vec{n}$.

3.4 Optimization Algorithm

The sequential quadratic programming method, SQP, is used to compute the descent direction and Lagrange multipliers ($\vec{v}$, λ) using information from our shape calculus analysis.

The Lagrangian function for the unconstrained problem is presented below.

$$\mathcal{L}\big(\Omega, \lambda\big) = J(\Omega) + \lambda\, C \tag{3.96}$$

where $C = |\Omega_b| - A_o = \int_{\Omega_b} dx - A_o$.

From Eqns. 3.94 and 3.48, shape derivative of the Lagrangian and derivative of the Lagrangian with respect to Lagrange multiplier are

$$\delta\mathcal{L}\big(\Omega; \vec{v}\big) = \delta J(\Omega; \vec{v}) + \lambda\, \delta C \tag{3.97}$$

$$\nabla_\lambda \mathcal{L} = C \tag{3.98}$$

From Eqn. 2.7, the Hadamard form of Eqn. 3.97 is

$$\begin{aligned} \langle \nabla\mathcal{L}\, \vec{n}, \vec{v}\rangle_{\Gamma_s} &= \langle \nabla J\, \vec{n}, \vec{v}\rangle_{\Gamma_s} + \lambda\, \langle \nabla C\, \vec{n}, \vec{v}\rangle_{\Gamma_s} \\ &= \big\langle (\nabla J + \lambda\, \nabla C), v\big\rangle_{\Gamma_s} \qquad (v = \vec{v}\cdot\vec{n}) \end{aligned} \tag{3.99}$$

Therefore,

$$\nabla\mathcal{L}(\Omega) = \nabla J(\Omega) + \lambda\, \nabla C(\Omega) \tag{3.100}$$

where

$$\nabla J\big(\Omega\big) = -2\mu\, \varepsilon(\vec{z}) : \varepsilon(\vec{u}) \tag{3.101}$$

$$\nabla C = 1 \tag{3.102}$$

The first order optimality or KKT conditions for Eqn. 3.96 are therefore

$$\mathcal{F}\big(\Omega, \lambda\big) = \begin{pmatrix} \nabla\mathcal{L}(\Omega) \\ \nabla_\lambda \mathcal{L} \end{pmatrix} = 0 \tag{3.103}$$

3.4.1 Newton's Method and SQP

Newton's method may be used to solve Eqn. 3.103. Starting from iterates v^k and λ^k, the converged solution for the descent direction and Lagrange multiplier are obtained by solving the following Newton's equation.

$$\begin{bmatrix} \nabla^2\mathcal{L}(\Omega^k) & \nabla C^k \\ \nabla C^k & 0 \end{bmatrix} \begin{pmatrix} d^k \\ s_\lambda^k \end{pmatrix} = \begin{pmatrix} -\nabla J^k - \lambda \nabla C^k \\ -C^k \end{pmatrix} \tag{3.104}$$

$\nabla^2\mathcal{L}$ is the Hessian of $\mathcal{L}$. Ω^{k+1} and λ^{k+1} are updated accordingly as

$$\Omega^{k+1} = \Omega^k + d^k\vec{n} \tag{3.105}$$

$$\lambda^{k+1} = \lambda^k + s_\lambda^k \tag{3.106}$$

From the theory of SQP methods, if we construct the equivalent quadratic model, the KKT condition for the quadratic model is analogous to solving the following set of equations

$$\begin{bmatrix} \nabla^2\mathcal{L}(\Omega^k) & \nabla C^k \\ \nabla C^k & 0 \end{bmatrix} \begin{pmatrix} d^k \\ \lambda^{k+1} \end{pmatrix} = \begin{pmatrix} -\nabla J^k \\ -C^k \end{pmatrix} \tag{3.107}$$

Eqn. 3.107 above is obtained by substituting s_λ^k using Eqn. 3.106

d^k as obtained from Eqn. 3.107 is defined on Γ_s and depends on the state and adjoint variables, $\vec{u}$, p and $\vec{z}$, r respectively. Even though both state and adjoint variables are defined everywhere on Ω, d^k may lack smoothness especially around Γ_s so that deforming the object using $\vec{v}^k = d^k\,\vec{n}$ computed directly from Eqn. 3.101 may introduce numerical instabilities. To avoid numerical problems, we solve Eqn. 3.108 which is the variational form equivalent of Eqn. 3.107. Eqn. 3.108 regularizes the decent direction by extending it over Ω and replacing the Hessian by an inner form for the shape variation. Although this regularized descent direction, $\vec{v}_{ex}$, is defined everywhere on Ω, it's numerical values are smoothened, especially on Γ_s. Thus, allowing for smooth shape deformation as drag is minimized. In this project,

we choose $\vec{v}_{ex}$ to be the unique solution to the following variational problem. For $\lambda \in R$ and $\vec{v}_{ex} \in R^d$ and for test functions $q \in R$ and $\vec{w} \in R^d$

$$\begin{aligned} \langle \vec{w}, \vec{v}_{ex} \rangle_\Omega + \langle \vec{w}, \lambda \nabla C \, \vec{n} \rangle_{\Gamma_s} &= -\langle \nabla J \, \vec{n}, \vec{w} \rangle_{\Gamma_s} \\ \langle q \nabla C \, \vec{n}, \vec{v}_{ex} \rangle_{\Gamma_s} &= -\langle C, q \rangle_{\Gamma_s} \end{aligned} \tag{3.108}$$

We have used the following expressing for $\langle \vec{w}, \vec{v}_{ex} \rangle_\Omega$.

$$\langle \vec{w}, \vec{v}_{ex} \rangle_\Omega = \int_\Omega \left(\nabla \vec{w} : \nabla \vec{v}_{ex} + \vec{w} \cdot \vec{v}_{ex} \right) d\Omega \tag{3.109}$$

Finite element method will be used to solve the above equation to obtain $\vec{v}_{ex}$. Indeed, Eqn. 3.108 guarantees that $\vec{v}_{ex}$ is a descent direction since the Lagrangian shape derivative and derivative of the Lagrangian with respect to the multiplier are negative. Other variational forms for the inner product for shape variation can be found in Burger (2003).

3.4.2 Algorithm for Numerical Optimization of Fluid Drag

Our Newton based numerical optimization algorithm proceeds as follows. For a given initial shape Γ_s and mesh to explicitly discretize Ω, for $k = 0$

1. Compute state variables, $\vec{u}$ and p by solving Navier-Stokes equation, Eqn. 3.16.
2. Compute fluid drag, J, according to Eqn. 3.1.
3. Compute adjoint variables, $\vec{z}$ and r by solving adjoint Navier-Stokes equation, Eqn. 3.51.
4. Compute regularized descent direction, $\vec{v}_{ex}$, by solving Eqn. 3.108.
5. Execute line search algorithm to find step length, α^{k+1}, along descent direction
6. Update shape, Ω^{k+1}, by moving coordinates of boundary nodes according to $\vec{x}^{k+1} = \vec{x}^k + \alpha^{k+1} \, \vec{v}_{ex}$.
7. If $|J^{k+1} - J^k| \leq \varepsilon$, end optimization algorithm. Return Ω^k.

8. Otherwise $k = k + 1$ and return to step 1.

3.4.3 Line Search with Merit Function

To balance reduction in fluid drag while maintaining area constraint of the obstacle, we use a line search algorithm based on Armijo rule. The line search is equipped with backtracking to determine a suitable step length along the descent direction that maximizes reduction in cost function. To guarantee sufficient decrease in the fluid drag, our line search is based on a merit function instead of the Lagrangian function of Eqn. 3.90. Our merit function is the exact penalty function given below.

$$\phi\left(\Omega, \rho\right) = J(\Omega) + \rho|C| \tag{3.110}$$

ρ is the penalty parameter. For a step size α^k to be accepted at the k^{th} optimization step, the Armijo criterion applied to Eqn. 3.110 requires that

$$\phi\left(\Omega^k + \alpha^k\, \vec{v}^k, \rho^k\right) \leq \phi\left(\Omega^k, \rho^k\right) + \eta\alpha^k \delta\phi\left(\Omega^k, \rho^k; \vec{v}^k\right), \qquad \eta \in (0, 1) \tag{3.111}$$

The expression for $\delta\phi(\Omega^k, \rho^k; \vec{v}^k)$ which is the shape derivative of ϕ along $\vec{v}^k$ is

$$\begin{aligned} \delta\phi\left(\Omega^k, \rho^k; \vec{v}^k\right) &= \delta J(\Omega^k; \vec{v}^k) - \rho^k|C^k| \\ &= -\langle \vec{v}^k, \vec{v}^k \rangle_{\Gamma_s} - \rho^k|C^k| \end{aligned} \tag{3.112}$$

(See Nocedal and Wright (2006), Biegler (2010) for derivation of Eqn. 3.112)

Selection of ρ^k is guided by the fact that $\vec{v}^k$ is guaranteed to be a descent direction for ϕ provided that $\rho^k \geq \lambda^k$ (Nocedal and Wright (2006)). Putting all together, our line search algorithm is summarized as follows:

Given Ω^k, $\vec{v}^k = \vec{v}^k_{ex}$, λ^k,ρ^{k-1} as inputs

1. Update penalty parameter $\rho^k = \max\left(\rho^{k-1}, \lambda^k\right)$

2. Compute $\phi\left(\Omega^k, \rho^k\right)$ by Eqn. 3.110.

3. Compute $\delta\phi\left(\Omega^k, \rho^k; \vec{v}^k\right)$ by Eqn. 3.112.

4. Set α^k to initial value i.e. $\alpha^k = \alpha_o$. Choose value for η. ($\eta = 0.0001$ for example).

5. Update domain by $\Omega^* = \Omega^k + \alpha^k \vec{v}^k$.

6. Compute $\phi\left(\Omega^*, \rho^k\right)$ by Eqn. 3.110.

7. If $\phi\left(\Omega^*, \rho^k\right) \leq \phi\left(\Omega^k, \rho^k\right) + \eta\alpha^k\delta\phi\left(\Omega^k, \rho^k; \vec{v}^k\right)$, end line search algorithm. Return α^k and ρ^k.

8. Otherwise, update step size by $\alpha^k = \alpha^k/2$ and return to step 5.

3.4.4 Numerical Implementation

All the partial differential equations (Navier-Stokes and adjoint Navier-Stokes) in this project are solved using the finite element method. Numerical implementation of the finite element method for solution of these equations is achieved using the python interface of the open source finite element package, FEniCS (FEniCS 2003; Logg, Mardal, and Wells 2012). Our optimization algorithm and other associated programs are also written in python.

The obstacle is deformed by moving its boundary nodes along the direction of $\vec{v}_{ex}$. To smoothen the overall mesh structure, all the other nodes of the computational mesh are also moved. The new coordinates of all nodes are obtained by solving an elasticity equation with boundaries given by coordinates of the new computational domain after deformation. Mathematically, the elasticity equation is

$$-\nabla \cdot \sigma\left(\vec{w}\right) = 0 \qquad \text{in } \Omega \tag{3.113}$$

$$\vec{w} = \Omega^k + \alpha^k \vec{v}_{ex}^k \qquad \text{on } \Gamma_s \tag{3.114}$$

$$\vec{w} = 0 \qquad \text{on } \Gamma_+ \cup \Gamma_- \cup \Gamma_{in} \cup \Gamma_{out} \tag{3.115}$$

Where

$$\sigma = 2\mu\,\varepsilon(\vec{w}) + \lambda(\nabla \cdot \vec{w})\mathbb{1} \tag{3.116}$$

E and ν are Youngs modulus and Poissons ratio respectively. μ and λ are the Lamé coefficients given by

$$\begin{aligned} \mu &= \frac{E}{2(1+\nu)} \\ \lambda &= \frac{\nu E}{(1+\nu)(1-2\nu)} \end{aligned} \tag{3.117}$$

The boundary condition of Eqn. 3.115 is due to the fact that boundaries other than Γ_s do not move during the optimization process. It is important to note that mesh smoothening using the elasticity equation does not guarantee continuous maintenance of good quality mesh during obstacle deformation. Mesh quality inevitably degenerates over time and we do periodic remeshing using the Triangle mesh generator (Shewchuk 1996a; Shewchuk 1996b) to improve mesh quality. In addition, FEniCS' automated goal-oriented error control is used to solve the Navier-Stokes equations to refine mesh around obstacle. However, a combination of remeshing and goal-oriented error control are not enough to prevent the obstacle from blowing out of the computational domain if the initial step size (α_o) is too large. Thus, α_o is carefully chosen to avoid advecting the boundaries of Γ_s beyond the external boundaries of the computational domain.

Chapter 4
Computational Results

4.1 Verification of Navier-Stokes Flow Solver

Our Navier-Stokes solver developed using FEniCS is first verified to ensure that it properly solves the state equations. The verification case is the 2D stationary flow benchmark computations in Schäfer et al. (1996). The problem geometry is shown in Fig. 4.1. It consists of a cylinder of diameter, $D = 0.1$m, centered at $x = 0.2$ m and $y = 0.2$ m in a rectangular domain of height, $H = 0.41$ m and length, $L = 2.2$ m.

Using our FEniCS code, fluid flow governed by the Navier-Stokes equation is simulated in the computational domain. From the numerically obtained flow variables, fluid drag around the obstacle, length of recirculation and pressure difference between the obstacle faces are computed and compared with values provided in Schäfer et al. (1996).

Following the approach of Schäfer et al. (1996), the inflow condition is

$$\vec{u}(0, y) = \big(4U_m y(H - y)/H^2, 0\big) = (u_x, u_y) \tag{4.1}$$

Reynold's number is defined as

$$Re = \rho \bar{U} D/\mu \tag{4.2}$$

$\bar{U}$ is mean velocity given by $= 2u_x(0, H/2)/3$. The verification case uses $U_m = 0.3$, $\rho = 1$ and $\mu = 0.001$, yielding $Re = 20$.

$$c_D = \frac{2F_D}{\rho \bar{U}^2 D} \tag{4.3}$$

Fluid drag, $F_D = J$, is computed using Eqn. 3.1 while drag coefficient is calculated using Eqn. 4.3. The length of recirculation is $L_a = x_r - x_e$, where $x_e = 0.25$ is the x-coordinate of the end of the cylinder and x_r is the x-coordinate of the end of the recirculation area. The pressure difference is defined as $\Delta p = p(x_a, y_a) - p(x_e, y_e)$, where $(x_a, y_a) = (0.15, 0.2)$ and $(x_e, y_e) = (0.25, 0.2)$ are the front and end points of the cylinder respectively.

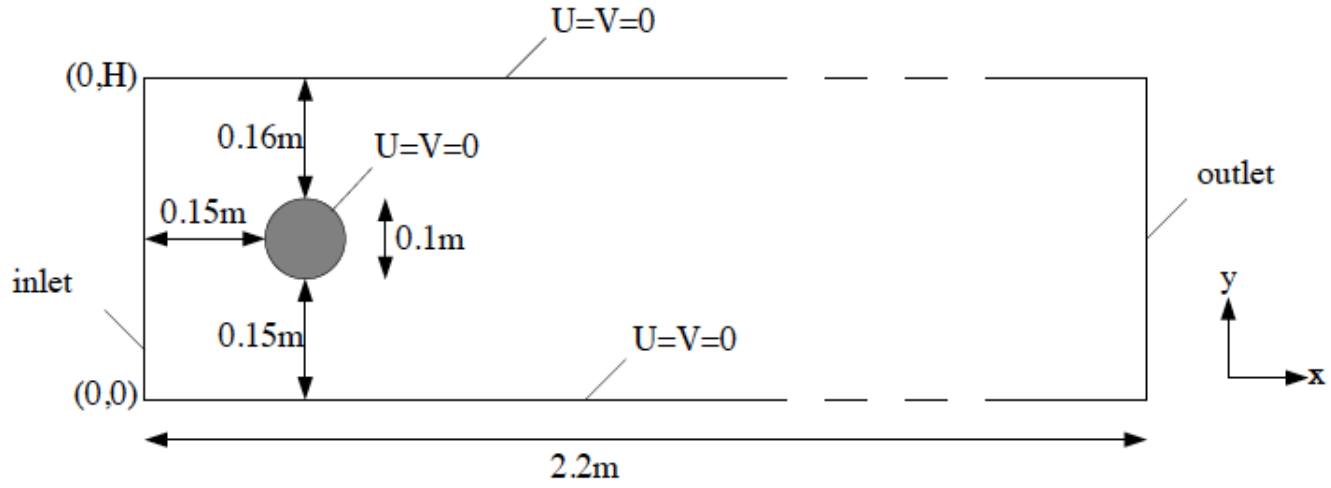

Figure 4.1: 2D geometry for Navier-Stokes solver verification case.

Fluid velocity and pressure from our FEniCS solver are compared with those from Margonari (2013), obtained using Scilab which is an open source software package. The results are shown in Figs. 4.3 and 4.4 respectively. Good comparison is obtained between both sets of results. Schäfer et al. (1996) reports values for drag coefficient, pressure difference and recirculation lengths for this test case as 5.5567, 0.0845 and 0.1172 respectively. We obtained a fluid drag of $J = 0.011$, yielding a drag coefficient of $c_D = 5.4922$. Fig. 4.2 shows our plot of fluid pressure and x-component of fluid velocity on a line through the cylinder to highlight pressure difference between the front and back of the obstacle and the fluid recirculation region respectively. From the left plot of Fig. 4.2, $\Delta p = 0.116361 - 0.0137084 = 0.1027$ while the right plot shows the recirculation region extends from 0.25 to 0.3325, giving a recirculation length of 0.0825. Our values are comparable to those in the Benchmark case of Schäfer et al. (1996).

4.2 Numerical Examples for Drag Minimization

We carry out computations for drag minimization in this section. The computational domain is shown in Fig. 4.5. It consists of a square obstacle placed in a rectangular flow domain. The inlet flow velocity is given by Eqn. 4.1, where $L = 21$m and $H = 7$m. Geometric constraint for the problem requires that the optimum shape has an area of 3.5m^2 i.e $A_o = 3.5\text{m}^2$. Clearly, the initial area of the square body is less than the area requirement

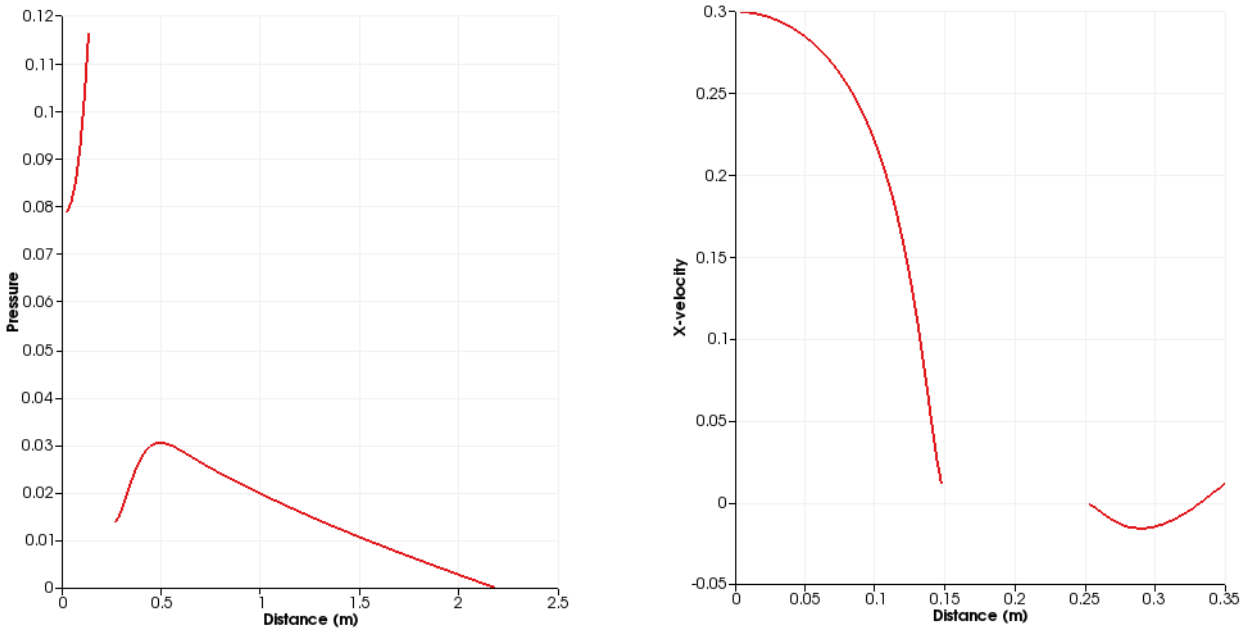

Figure 4.2: Left figure is fluid pressure plot along line on y = 0.2 extending from domain inlet to x = 2.5. It shows pressure difference between the front and back of obstacle in the flow domain. Right figure is plot of x-component of velocity along line on y = 0.2 extending from inlet to x = 0.35. The figure shows recirculation occurs between x ≈ 0.25-0.3325.

of the final shape. Input flow parameters are U_m and μ. For Reynold's number calculation, we assume obstacle diameter is equal to the length of the initial square i.e. $D = 1.2$. Therefore, by using different combinations of U_m and μ, we are able to carry out computations for different Reynold's numbers. Computations for five different Reynold's numbers were carried out and the combinations of U_m and μ are shown in Table 4.1. In addition, α_o is varied between computations to prevent Γ_s from advecting beyond the boundaries of the rectangle through out the optimization steps. Our FeniCS-Python code developed to implement the shape optimization algorithm discussed in Chapter 3 are shown in Appendix B and C. Convergence in our numerical computations is achieved when numerical error given by $|J^{k+1} - J^k|$ is less than some pre-selected tolerance value. We carried out the optimization step in two steps. In the first step, a coarse mesh is used and computation is carried out until numerical convergence is achieved. Thereafter, the optimization step is repeated with a refined mesh obtained from goal-oriented error control of the Navier-Stokes solver. Our

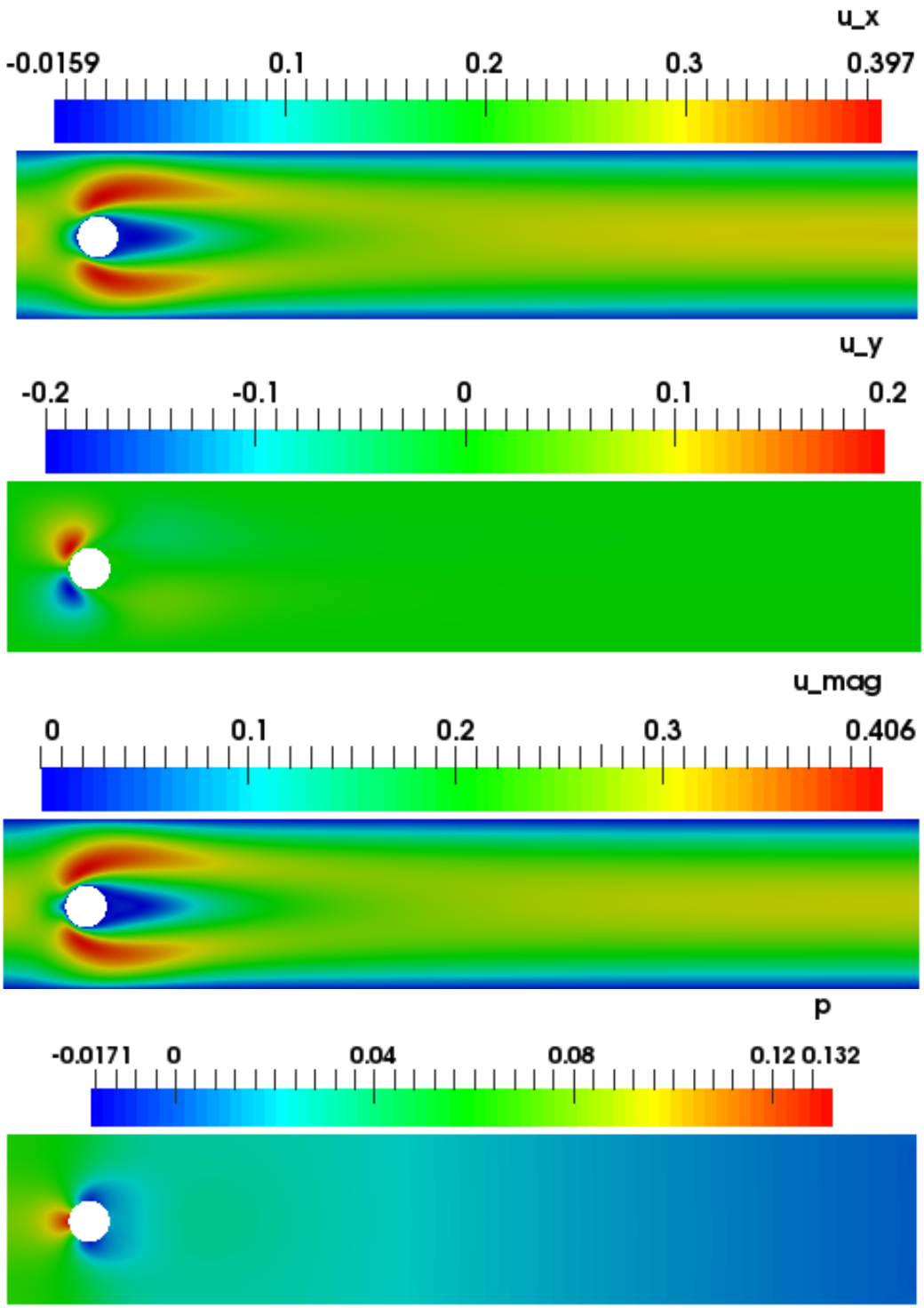

Figure 4.3: Flow results from our FEniCS solver for verification case. From top to bottom are x and y components of velocity, velocity magnitude and fluid pressure for the verification case.

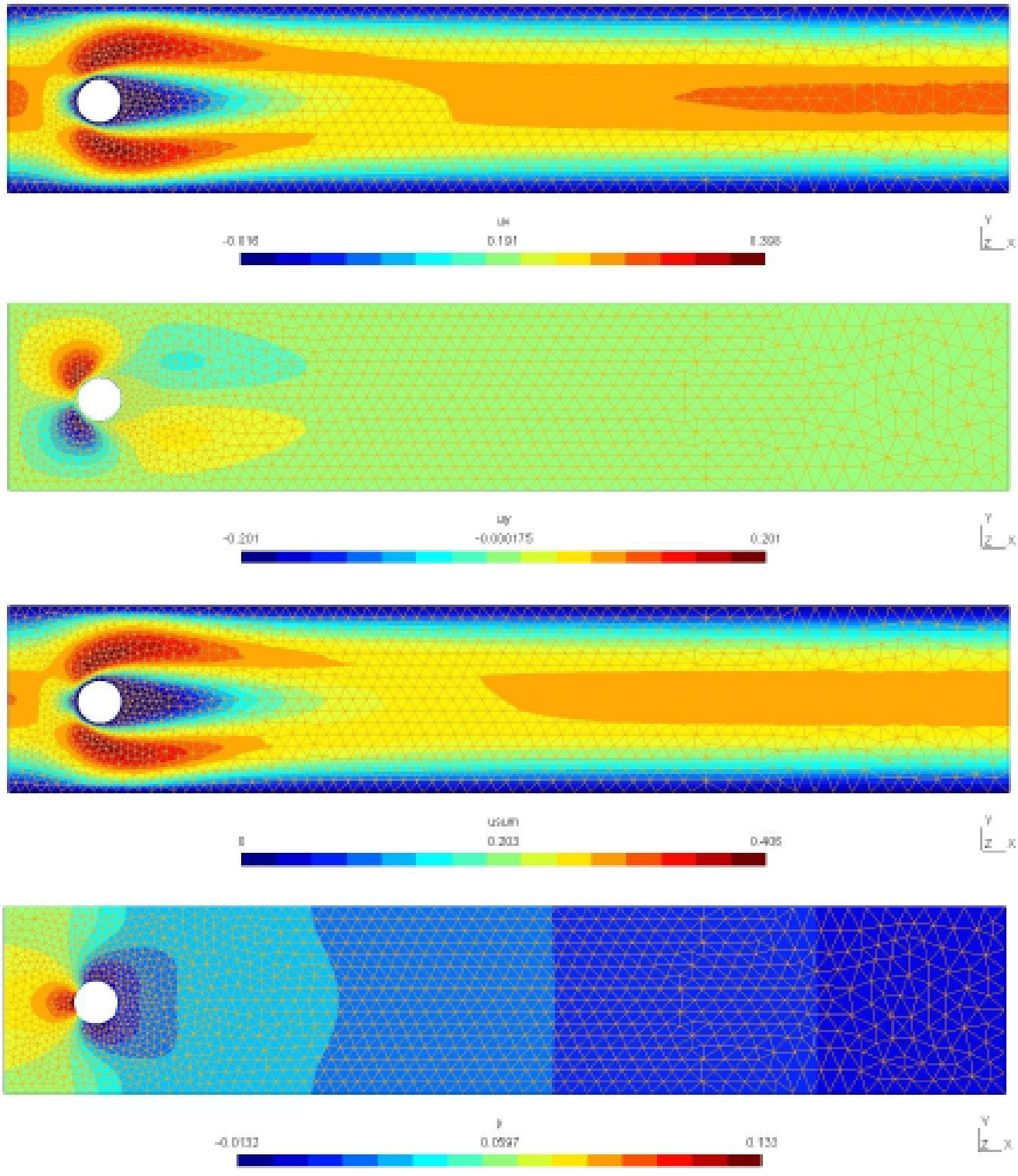

Figure 4.4: Flow results from Margonari (2013) for verification case. From top to bottom are x and y components of velocity, velocity magnitude and fluid pressure for the verification case.

U_m	μ	Re
0.125	1.0	0.1
0.5	0.4	1
2.5	0.05	40
5.0	0.04	100
5.0	0.02	200

Table 4.1: Input flow parameters for shape optimization problem.

numerical results for optimal shapes at different Re are shown in Figs. 4.7, 4.8 and 4.6. Fig. 4.7 shows velocity magnitude in the computational domain containing the object at final shape. It The final shapes for low Re's ($Re = 0.1$ and 1.0) are basically the same, symmetric in the $x-$ and $y-$directions and resemble the shape of an american football. This result has been obtained by several authors including Pironneau (1973), Morin et al. (2011), Montenegro-Johnson and Lauga (2015), Lindemann et al. (2012). At higher Re's however, the final shape is only symmetric in the $y-$direction as it has a more streamlined structure at the tail end than at the head section. In Fig. 4.8, we zoom into the computational domain to provide clearer details about final shapes at different Re's. Fig. 4.6 highlights evolution of the object's shape starting from initial to final, for $Re = 200$. It is obvious that the final shape enhances flow as recirculation is eliminated due to the streamlined shape of the object. Finally, Figs. 4.9 and 4.10 provide some information about the computations. Clearly, the geometric constraint given by $C := |\Omega_b| - 3.5 \approx 0.0$ is met for all computations, where $|\Omega_b| = L \times H - \int_\Omega dx$. The small increments in values of J, $\mathcal{L}$ and ϕ in Fig. 4.9 during the optimization steps are the result periodic remeshing to improve mesh quality.

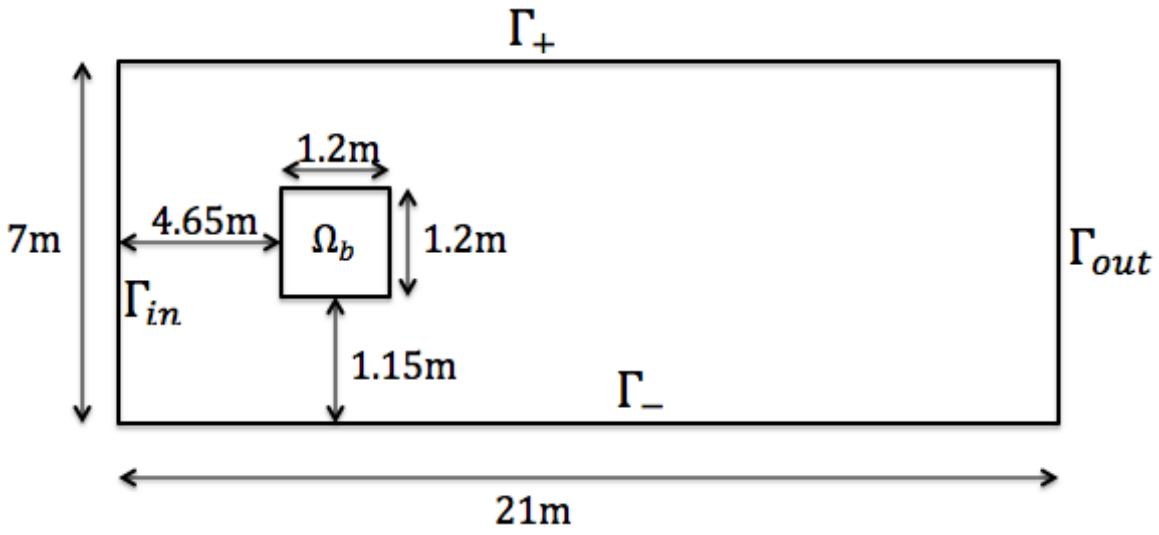

Figure 4.5: Initial domain for drag minimization computations.

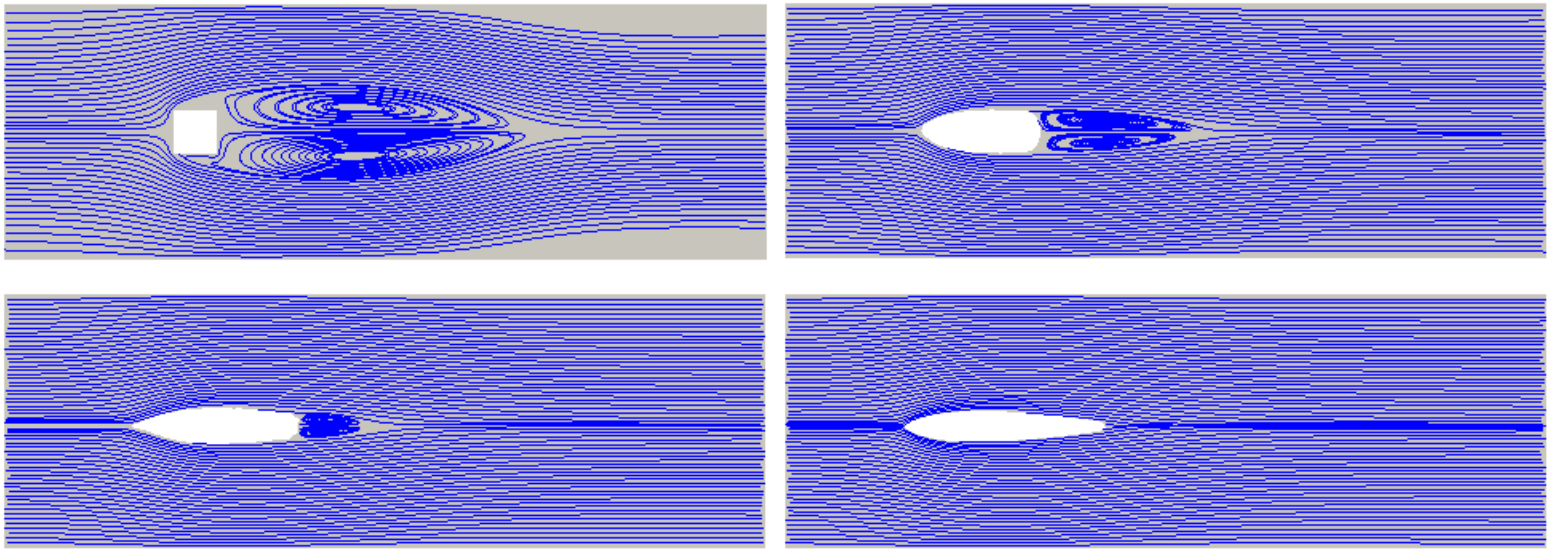

Figure 4.6: Fluid streamlines around the obstacle at different optimization steps for $Re = 200$. Top left figure shows streamlines at initial step. Top right, bottom left and bottom right are streamlines around object after 10, 40 and final optimization steps respectively. The final shape ensures that fluid vortices are eliminated around the object.

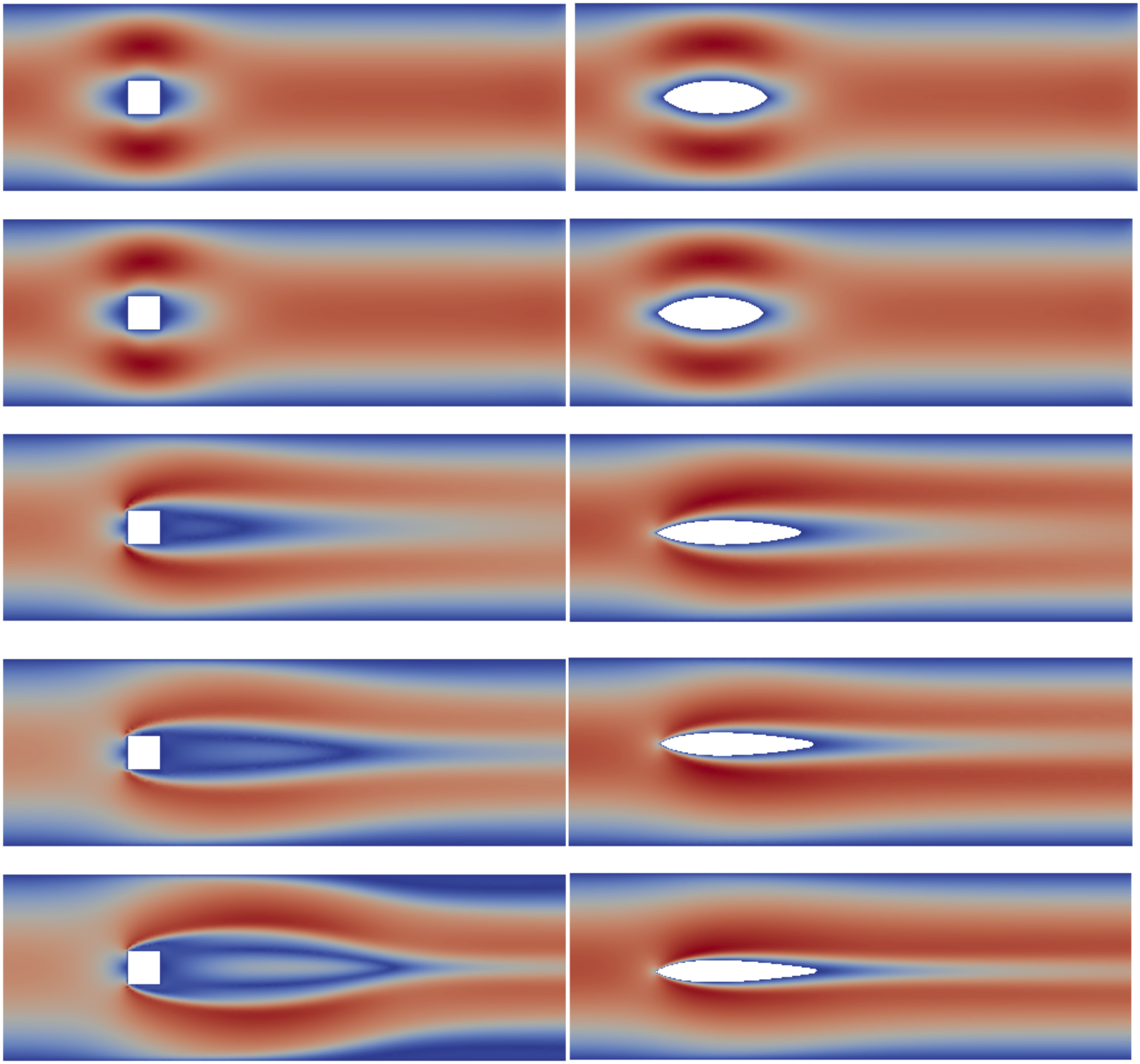

Figure 4.7: Velocity magnitude around in initial domain and final computational domain containing optimum shape that minimizes least drag for different Reynolds numbers. Left figures show velocity magnitude in initial domains while figures on the right are velocity magnitude in computational domain containing optimal profiles. From top to bottom, $Re = 0.1, 1, 40, 100$ and 200.

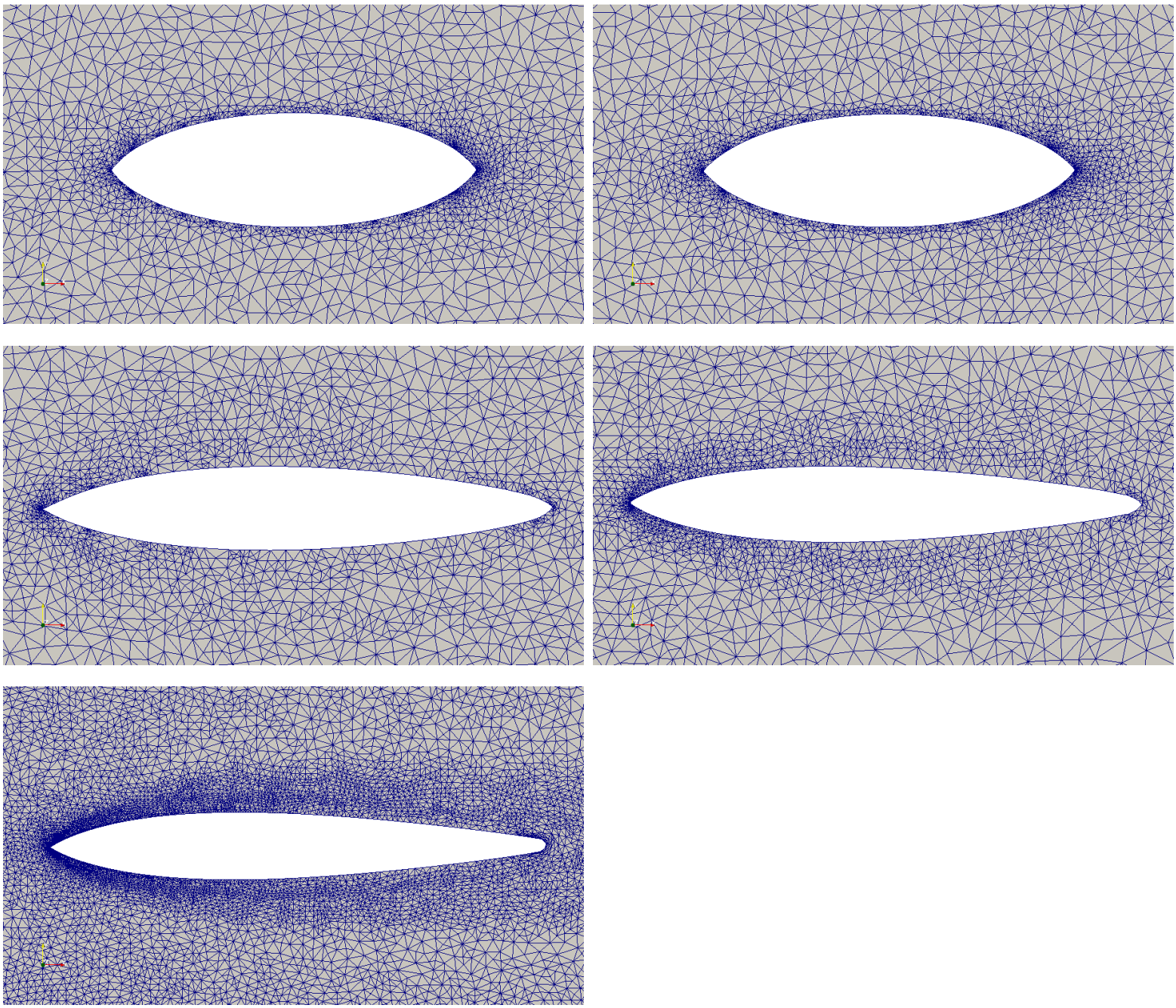

Figure 4.8: Magnified image of final object shapes for different Reynolds numbers. Figures from top to bottom are $Re = 0.1, 1, 40, 100$ and 200.

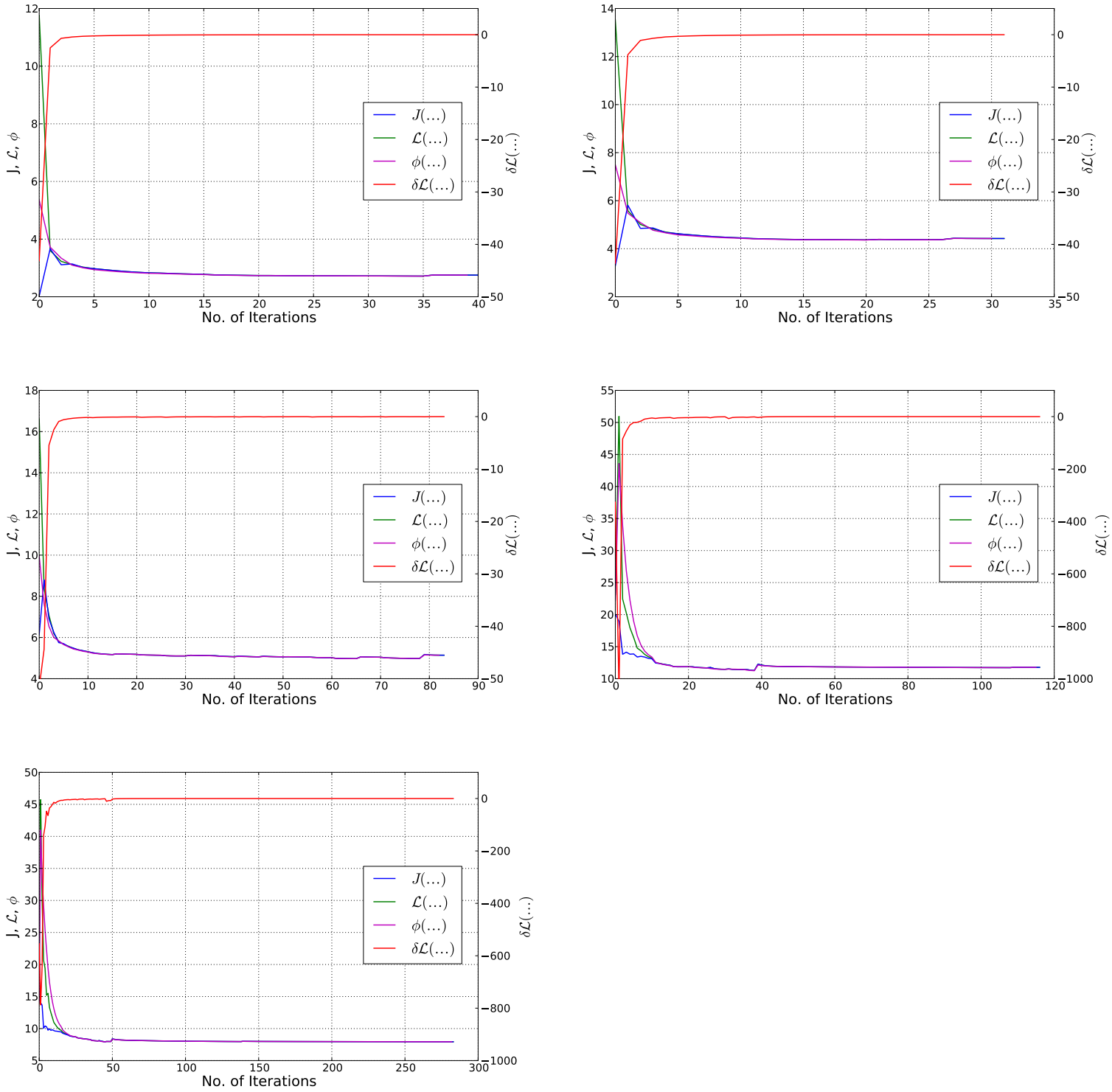

Figure 4.9: Evolution of J, $\mathcal{L}$, ϕ and $\delta\mathcal{L}$ for different Reynolds numbers. Plots from top to bottom are for $Re = 0.1, 1, 40, 100$ and 200. The values of J, $\mathcal{L}$ and ϕ are different at early optimization steps since geometric constraint is not satisfied. As optimization progresses, drag is minimized and geometric constraint is satisfied. As a result, $J \approx \mathcal{L} \approx \phi$. $\delta\mathcal{L}$ has negative values during optimization since $\mathcal{L}$ is reducing. $|\partial\mathcal{L}| \approx 0$ at the end of optimization.

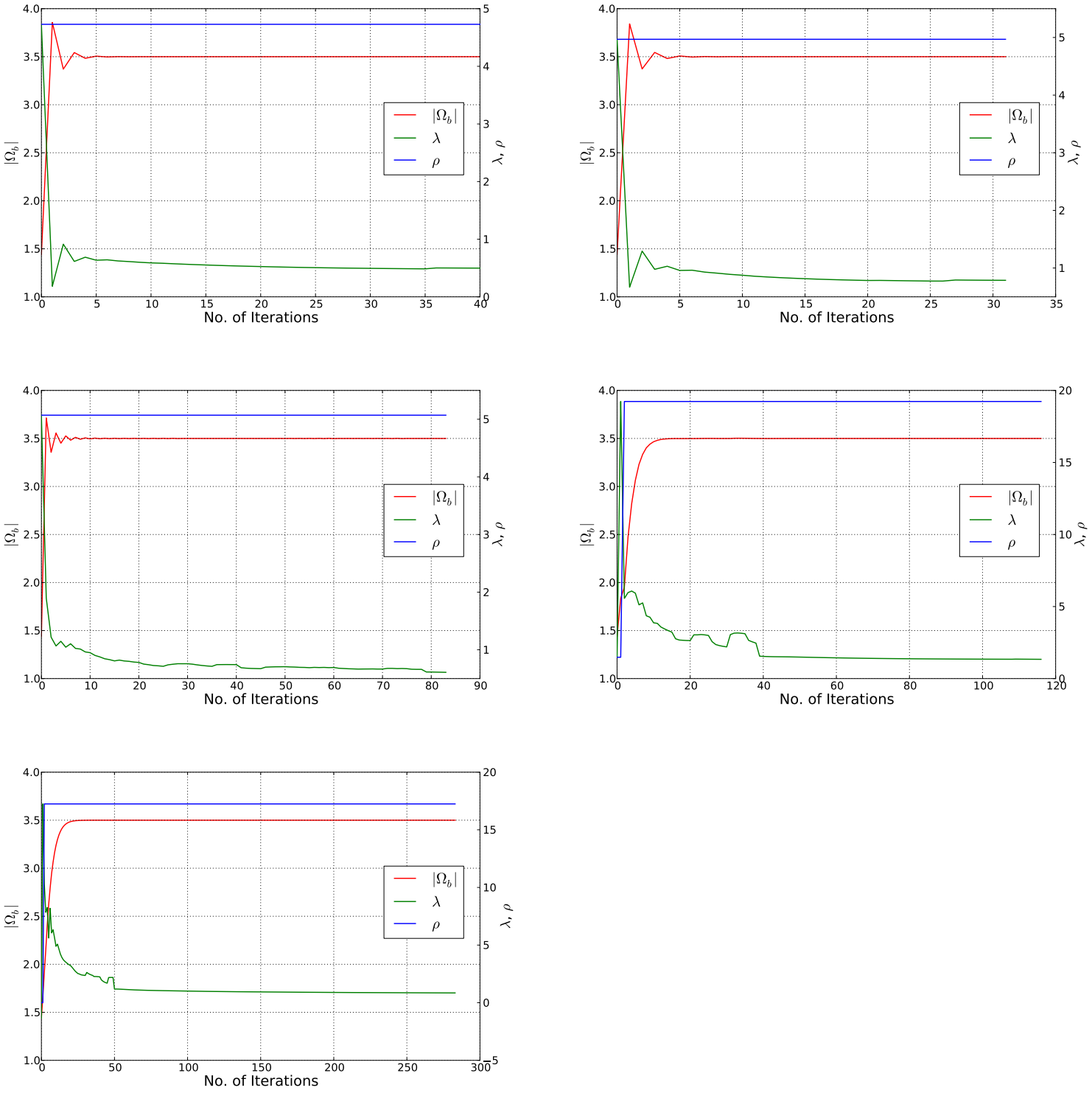

Figure 4.10: Evolution of $|\Omega_b|$, λ and ρ during optimization steps for different Reynolds numbers. Plots from top to bottom are for $Re = 0.1, 1, 40, 100$ and 200. It is clearly observed that although initial obstacle area is less than the area requirement, geometric constraint is met for all computations. We also observe that $\rho \geq \lambda$.

Chapter 5
Conclusion

We have solved the drag minimization problem involving a stationary object placed in a flowing channel. Fluid flow was modeled by Navier-Stokes equation while geometric constraint involved a specified area for the stationary object. The geometric constraint was removed using the Lagrangian method and shape sensitivity analysis applied to the Lagrangian generated the adjoint Navier-Stokes equation, shape derivative and corresponding shape gradient. With the shape gradient as descent direction, the sequential quadratic programming (SQP) technique was used as the optimization method. The variational equivalent form of the SQP method was solved with the Hessian of the Lagrangian replaced by an inner product for shape variation to guarantee smooth shape deformation. Mesh deformation was carried out by moving the nodes of the object boundaries. The optimization algorithm was implemented in Python. We carried out 2D numerical computations at different Reynold's numbers for the optimal shapes that minimize drag. Shapes at low Reynold's number were symmetric in both x and y directions while moderate to high Reynold's numbers had shapes that were more streamlined at the front of the object. Computation results also showed gradual decrease in drag Lagrangian. In addition, the geometric constraints were met for all computations. All numerical algorithm were coded in Python while PDE's were solved using the python interface of the open source software, FEniCS. Triangle mesh generator was used for meshing the computational domain.

References

Biegler, Lorenz T. 2010. *Nonlinear programming: concepts, algorithms, and applications to chemical processes*. Volume 10. SIAM.

Brandenburg, Christian, Florian Lindemann, Michael Ulbrich, and Stefan Ulbrich. 2009. "A continuous adjoint approach to shape optimization for Navier Stokes flow." In *Optimal control of coupled systems of partial differential equations*, 35–56. Springer.

Burger, Martin. 2003. "A framework for the construction of level set methods for shape optimization and reconstruction." *Interfaces and Free boundaries* 5 (3): 301–330.

Dede, L. 2007. "Optimal flow control for Navier–Stokes equations: drag minimization." *International Journal for Numerical Methods in Fluids* 55 (4): 347–366.

Delfour, MC, and JP Zolésio. 2001. "Shapes and Geometries–Analysis, Differential Calculus and Optimization, Adv." *Design and Control, SIAM*.

FEniCS. 2003. FEniCS Project. `http://fenicsproject.org/`.

Hadamard, Jacques. 1908. *Mémoire sur le problème d'analyse relatif à l'équilibre des plaques élastiques encastrées*. Volume 33. National Printing.

Haslinger, Jaroslav, et al. 2003. *Introduction to shape optimization: theory, approximation, and computation*. Volume 7. SIAM.

Lindemann, Florian, et al. 2012. "Theoretical and numerical aspects of shape optimization with Navier-Stokes flows." Ph.D. diss., Technische Universität München.

Logg, Anders, Kent-Andre Mardal, and Garth Wells. 2012. *Automated solution of differential equations by the finite element method: The FEniCS book*. Volume 84. Springer Science & Business Media.

Margonari, Massimiliano. 2013. "SCILAB FINITE ELEMENT SOLVER."

Montenegro-Johnson, Thomas D, and Eric Lauga. 2015. "The other optimal Stokes drag profile." *Journal of Fluid Mechanics* 762:R3.

Morin, Pedro, Ricardo H Nochetto, M Sebastian Pauletti, and Marco Verani. 2011. "AFEM for shape optimization." *Report MOX*, vol. 29.

Nocedal, Jorge, and Stephen Wright. 2006. *Numerical optimization*. Springer Science & Business Media.

Pironneau, Olivier. 1973. "On optimum profiles in Stokes flow." *Journal of Fluid Mechanics* 59 (01): 117–128.

———. 2012. *Optimal shape design for elliptic systems*. Springer Science & Business Media.

Schäfer, Michael, Stefan Turek, F Durst, E Krause, and R Rannacher. 1996. *Benchmark computations of laminar flow around a cylinder*. Springer.

Shewchuk, Jonathan. 1996a. Triangle. `https://www.cs.cmu.edu/ quake/triangle.html`.

Shewchuk, Jonathan Richard. 1996b. "Triangle: Engineering a 2D quality mesh generator and Delaunay triangulator." In *Applied computational geometry towards geometric engineering*, 203–222. Springer.

Slawig, Thomas. 2003. "A Formula for the Derivative with Respect to Domain Variations in Navier–Stokes Flow Based on an Embedding Domain Method." *SIAM journal on control and optimization* 42 (2): 495–512.

———. 2006. "PDE-constrained control using FEMLAB–Control of the Navier–Stokes equations." *Numerical Algorithms* 42 (2): 107–126.

Walker, Shawn W. 2015. *The Shapes of Things: A Practical Guide to Differential Geometry and the Shape Derivative*. Volume 28. SIAM.

Zolésio, J Sokolowski-JP. 1992. "Introduction to Shape Optimization: Shape Sensitivity Analysis." *Springer Ser. Comput. Math*, vol. 10.

Appendix A
Analysis of Terms in Material Derivative of Drag Lagrangian

Analysis of $a(\vec{u}', \vec{z})$

$a(\vec{u}', \vec{z})$ is obtained by deriving the weak formulation of the adjoint equation using $\vec{u}'$ as the test function. Multiplying Eqn. 3.60 with $\vec{u}'$ and doing integration by parts, we have

$$\begin{aligned} -\int_\Omega \nabla \cdot T \cdot \vec{u}' + \int_\Omega (\nabla \vec{u}^T \cdot \vec{z})\, \vec{u}' - \int_\Omega (\vec{u} \cdot \nabla)\, \vec{z} \cdot \vec{u}' = 0 \\ \int_\Omega T : \nabla \vec{u}' - \int_{\partial\Omega} T \cdot \vec{n} \cdot \vec{u}' + \int_\Omega (\nabla \vec{u}^T \cdot \vec{z})\, \vec{u}' - \int_\Omega (\vec{u} \cdot \nabla)\, \vec{z} \cdot \vec{u}' = 0 \\ -\int_\Omega \vec{r} \nabla \cdot \vec{u}' + 2\mu \int_\Omega \varepsilon(\vec{z}) : \varepsilon(\vec{u}') - \int_{\partial\Omega} T \cdot \vec{n} \cdot \vec{u}' + \int_\Omega (\nabla \vec{u}^T \cdot \vec{z}) \vec{u}' - \int_\Omega (\vec{u} \cdot \nabla \vec{z}) \cdot \vec{u}' = 0 \end{aligned} \tag{A.1}$$

$T \cdot \vec{n} \cdot \vec{u}' = T \cdot \vec{u}' \cdot \vec{n}$ due to symmetry of T and $\nabla \cdot \vec{u} = 0$ from continuity equation. Therefore,

$$\begin{aligned} 2\mu \int_\Omega \varepsilon(\vec{z}) : \varepsilon(\vec{u}') &= \int_{\partial\Omega} T \cdot \vec{n} \cdot \vec{u}' - \int_\Omega (\nabla \vec{u}^T \cdot \vec{z}) \cdot \vec{u}' + \int_\Omega (\vec{u} \cdot \nabla)\, \vec{z} \cdot \vec{u}' \\ &= \int_{\Gamma_{in}} T \cdot \vec{n} \cdot \vec{u}' + \int_{\Gamma_+} T \cdot \vec{n} \cdot \vec{u}' + \int_{\Gamma_-} T \cdot \vec{n} \cdot \vec{u}' + \int_{\Gamma_s} T \cdot \vec{n} \cdot \vec{u}' + \int_{\Gamma_{out}} T \cdot \vec{n} \cdot \vec{u}' \\ &\quad - \int_\Omega (\nabla \vec{u}^T \cdot \vec{z}) \cdot \vec{u}' + \int_\Omega (\vec{u} \cdot \nabla)\, \vec{z} \cdot \vec{u}' \end{aligned} \tag{A.2}$$

Boundary integrals over Γ_{in}, Γ_{out}, Γ_+ and Γ_- are equal to zero since $\vec{u}' = 0$ on those boundaries. Thus,

$$a(\vec{u}', \vec{z}) = 2\mu \int_\Omega \varepsilon(\vec{z}) : \varepsilon(\vec{u}') = \int_{\Gamma_s} T \cdot \vec{n} \cdot \vec{u}' - \int_\Omega (\nabla \vec{u}^T \cdot \vec{z}) \cdot \vec{u}' + \int_\Omega (\vec{u} \cdot \nabla)\, \vec{z} \cdot \vec{u}' \tag{A.3}$$

Analysis of $a(\vec{u}, \vec{z}')$

$a(\vec{u}, \vec{z}')$ is also obtained by deriving the weak formulation of the state equation using $\vec{z}'$ as the test function. Multiplying Eqn. 3.2 with $\vec{z}'$ and doing integration by parts, we have

$$\begin{aligned} -\int_\Omega \nabla \cdot \sigma \cdot \vec{z}' + \int_\Omega (\vec{u} \cdot \nabla)\, \vec{u} \cdot \vec{z}' = 0 \\ \int_\Omega \sigma : \nabla \vec{z}' - \int_{\partial\Omega} \sigma \cdot \vec{n} \cdot \vec{z}' + \int_\Omega (\vec{u} \cdot \nabla \vec{u}) \cdot \vec{z}' = 0 \\ -\int_\Omega \vec{p} \nabla \cdot \vec{z}' + 2\mu \int_\Omega \varepsilon(\vec{u}) : \varepsilon(\vec{z}') - \int_{\partial\Omega} \sigma \cdot \vec{n} \cdot \vec{z}' + \int_\Omega (\vec{u} \cdot \nabla)\, \vec{u} \cdot \vec{z}' = 0 \end{aligned} \tag{A.4}$$

$\nabla \cdot \vec{z}' = 0$ from material derivative of adjoint equation. Therefore,

$$\begin{aligned} 2\mu \int_\Omega \varepsilon(\vec{u}) : \varepsilon(\vec{z}') &= \int_{\partial\Omega} \sigma \cdot \vec{n} \cdot \vec{z}' - \int_\Omega (\vec{u} \cdot \nabla)\, \vec{u} \cdot \vec{z}' \\ &= \int_{\Gamma_{in}} \sigma \cdot \vec{n} \cdot \vec{z}' + \int_{\Gamma_+} \sigma \cdot \vec{n} \cdot \vec{z}' + \int_{\Gamma_-} \sigma \cdot \vec{n} \cdot \vec{z}' + \int_{\Gamma_s} \sigma \cdot \vec{n} \cdot \vec{z}' + \int_{\Gamma_{out}} \sigma \cdot \vec{n} \cdot \vec{z}' \\ &\quad - \int_\Omega (\vec{u} \cdot \nabla)\, \vec{u} \cdot \vec{z}' \end{aligned} \tag{A.5}$$

In the equation above, the boundary integrals over Γ_{in}, Γ_+, Γ_- are zero because $\vec{z}' =$ on those boundaries. Integral over Γ_{out} is also zero because $\sigma \cdot \vec{n} = 0$. Thus,

$$a(\vec{u}, \vec{z}') = 2\mu \int_\Omega \varepsilon(\vec{u}) : \varepsilon(\vec{z}') = \int_{\Gamma_s} \sigma \cdot \vec{n} \cdot \vec{z}' - \int_\Omega (\vec{u} \cdot \nabla)\, \vec{u} \cdot \vec{z}' \tag{A.6}$$

Further Analysis I

Consider

$$\begin{aligned} \int_\Omega \nabla\Big((\vec{z} \cdot \vec{u}')\vec{u}\Big) &= \int_{\partial\Omega} (\vec{z} \cdot \vec{u}')(\vec{u} \cdot \vec{n}) \\ &= \int_{\Gamma_{in} \cup \Gamma_+ \cup \Gamma_- \cup \Gamma_s} (\vec{z} \cdot \vec{u}')(\vec{u} \cdot \vec{n}) + \int_{\Gamma_{out}} (\vec{z} \cdot \vec{u}')(\vec{u} \cdot \vec{n}) \end{aligned} \tag{A.7}$$

The first integral on the right hand side of the above equation is zero because $\vec{u} = 0$ on $\Gamma_+ \cup \Gamma_- \cup \Gamma_s$, $\vec{z} = 0$ on $\Gamma_{in} \cup \Gamma_+ \cup \Gamma_-$ and $\vec{u}' = 0$ on $\Gamma_{in} \cup \Gamma_+ \cup \Gamma_- \cup \Gamma_{out}$. Applying all of these,

$$\begin{aligned} 0 &= \int_\Omega \nabla\Big((\vec{z} \cdot \vec{u}')\vec{u}\Big) \\ &= \int_\Omega \vec{u} \cdot \nabla(\vec{z} \cdot \vec{u}') + \int_\Omega (\vec{z} \cdot \vec{u}')\nabla \cdot \vec{u} \qquad (\text{but } \nabla \cdot \vec{u} = 0) \\ &= \int_\Omega (\vec{u} \cdot \nabla \vec{z}) \cdot \vec{u}' + \int_\Omega (\vec{u} \cdot \nabla \vec{u}') \cdot \vec{z} \end{aligned} \tag{A.8}$$

Therefore

$$\int_\Omega (\vec{u} \cdot \nabla \vec{z}) \cdot \vec{u}' + \int_\Omega (\vec{u} \cdot \nabla \vec{u}') \cdot \vec{z} = 0 \tag{A.9}$$

Further Analysis II

Consider that $\vec{z}' = -\nabla \vec{z} \cdot \vec{V} = -\nabla \vec{z} \cdot \vec{n}\, V$, we have

$$\begin{aligned}\int_{\Gamma_s} \sigma \cdot \vec{n} \cdot \vec{z}' &= -\int_{\Gamma_s} (\sigma \cdot \vec{n})(\nabla \vec{z} \cdot \vec{n}) V \\ &= -\int_{\Gamma_s} \sigma_{ik} n_k z_{i,j} n_j V \quad (z_{i,j} = \frac{\partial z_i}{\partial x_j}) \\ &= -\int_{\Gamma_s} \sigma_{ik} (z_{i,j} n_j n_k) V \\ &= -\int_{\Gamma_s} \sigma : \big(\nabla \vec{z} (\vec{n} \otimes \vec{n})\big) V\end{aligned} \tag{A.10}$$

$\nabla \vec{z}$ can be decomposed into two components as shown below. $\nabla \vec{z}_\Gamma$ and $\nabla \vec{z}_n$ which are projections of $\nabla \vec{z}$ on the tangent plane and in the normal directions respectively.

$$\nabla \vec{z} = \nabla \vec{z}_\Gamma + \nabla \vec{z}_n \tag{A.11}$$

where $\nabla \vec{z}_n = \nabla \vec{z} (\vec{n} \otimes \vec{n})$. $\vec{a} \otimes \vec{b}$ is dyadic vector products given by $\vec{a} \otimes \vec{b} = a_i b_j$
Since $\vec{z} = \phi$ on Γ_s, it is a constant and $\nabla \vec{z}_\Gamma$ is therefore a zero tensor. As a result,

$$\nabla \vec{z} = \nabla \vec{z}_n = \nabla \vec{z} (\vec{n} \otimes \vec{n}) \tag{A.12}$$

Therefore, Eqn. A.10 becomes

$$\begin{aligned}\int_{\Gamma_s} \sigma \cdot \vec{n} \cdot \vec{z}' &= -\int_{\Gamma_s} \sigma : \nabla \vec{z} V \\ &= \int_{\Gamma_s} p : \nabla \cdot \vec{z} V - 2\mu \int_{\Gamma_s} \varepsilon(\vec{u}) : \varepsilon(\vec{z})\, V \quad (\nabla \cdot \vec{z} = 0) \\ &= -2\mu \int_{\Gamma_s} \varepsilon(\vec{u}) : \varepsilon(\vec{z})\, V\end{aligned} \tag{A.13}$$

Similar analysis can be done to show that

$$\int_{\Gamma_s} T \cdot \vec{n} \cdot \vec{u}' = -2\mu \int_{\Gamma_s} \varepsilon(\vec{u}) : \varepsilon(\vec{z})\, V \tag{A.14}$$

Appendix B
Optimization Code in Python

```
1  from dolfin import *
2  import sys, math, numpy
3  import functionsfile as funcs
4  import os
5  import time
6
7  if not has_cgal():
8    print "DOLFIN must be compiled with CGAL to run this demo."
9    exit(0)
10
11 lx = 21.
12 ly = 7.
13 cx = lx/4.
14 cy = ly/2.
15 circ_rad = 0.6
16 Ao = 3.5
17 tol = 1e-5
18 k = 0
19 l = 0
20 alpha = 1.0;
21 U_m = 0.125
22 rho = 1.0
23 nu = 0.1
24 mu = rho*nu
25 penalty_function = 0
26 penalty_parameter = 0
27 lam = 0
28
29 domain_r = Rectangle(0., 0, lx, ly)-Rectangle(cx-circ_rad, cy-circ_rad, cx+
       circ_rad, cy+circ_rad)
30 mesh = Mesh(domain_r, 100)
31 boundary_parts = funcs.MarkBoundaries(mesh,lx,ly)
32
33 Drag_o = 1e+3
34 NSBool = False
35 for i in range(2):
36   area_error = [1e+6]
37   drag_error = [1e+6]
38   if i == 0:
39     tol = 1e-6
40     error = area_error
41   else:
42     tol = 1e-3
43     error = drag_error
44     mesh, boundary_parts = funcs.CreateNewMeshUsingTriangle(mesh,lx,ly,
           boundary_parts)
45     p,u,mesh,boundary_parts = funcs.NSSolver(mesh,boundary_parts,lx,ly,mu,U_m,
           True)
46   while (error[0] > tol and drag_error[0] > 1e-6):
```

```
47      p,u,mesh,boundary_parts = funcs.NSSolver(mesh,boundary_parts,lx,ly,mu,U_m,
            NSBool)
48      r,z = funcs.AdjointSolver(mesh,boundary_parts,mu,u)
49      w,lam = funcs.SearchDirection(mesh,boundary_parts,mu,u,z,lx,ly, Ao)
50      Drag = funcs.ComputeDrag(mesh,boundary_parts,mu,p,u)
51      Der_Drag = funcs.DragLagrangianDerivative(mesh,boundary_parts,mu,u,z,w,lx,
            ly,lam,Ao)
52      Drag_Lagrangian = funcs.ComputeDragLagrangian(mesh,boundary_parts,mu,p,u,
            lx,ly,lam,Ao)
53      penalty_function,alpha,penalty_parameter,mesh,boundary_parts = funcs.
            ArmijoLineSearch(mesh,boundary_parts,w,mu,U_m,lx,ly,lam,
            penalty_parameter,Ao)
54      drag_error[0] = abs(Drag-Drag_o)
55      area_error[0] = abs(assemble(Constant(1.0) * dx, mesh=mesh)-(lx*ly-Ao))
56      Area = lx*ly-assemble(Constant(1.0) * dx, mesh=mesh)
57      Drag_o = Drag
58      if k % 5 == 0 and i == 0:
59        mesh, boundary_parts = funcs.CreateNewMeshUsingTriangle(mesh,lx,ly,
              boundary_parts)
60      k = k+1
```

Appendix C
FEniCS-Python Functions File

```
from dolfin import *
import sys, math, numpy
import os

E = 1
nu = 0.3
mu = E/(2*(1+nu))
lamda = E*nu/((1+nu)*(1-2.0*nu))

def epsilon(u):
    return 0.5*(grad(u) + grad(u).T)

def sigma(u):
    return 2*mu*epsilon(u)+lamda*tr(epsilon(u))*Identity(len(u))


def ComputeDrag(mesh,boundary_parts,mu,p,u):
    n_infnty = Constant((1, 0))
    n = FacetNormal(mesh)
    I = Identity(2)
    sigma = -p*I + 2*mu*epsilon(u)
    M1 = -1.*dot(dot(sigma,n),n_infnty)*ds(0)
    Drag = assemble(M1,exterior_facet_domains=boundary_parts)
    return Drag

def NSSolver(mesh,boundary_parts,lx,ly,mu,U_o,ShouldRefine):
    parameters['allow_extrapolation'] = True
    V = VectorFunctionSpace(mesh, "Lagrange", 2)
    Q = FunctionSpace(mesh, "CG", 1)
    W = V * Q
    noslip = Constant((0, 0))
    bc0 = DirichletBC(W.sub(0), noslip, boundary_parts, 3)
    bc1 = DirichletBC(W.sub(0), noslip, boundary_parts, 0)
    inflow = Expression(("4*u_in*x[1]*(H-x[1])/(H*H)", "0.0"),H=ly,u_in=U_o)
    bc2 = DirichletBC(W.sub(0), inflow, boundary_parts, 2)
    bcs = [bc0, bc1, bc2]
    (v, q) = TestFunctions(W)
    f = Constant((0, 0))
    w = Function(W)
    (u, p) = split(w)
    F = (2*mu*(inner(epsilon(u), epsilon(v))) - div(v)*p - q*div(u)+inner(grad(u
        )*u,v) )*dx-inner(f, v)*dx
    ds = Measure("ds")[boundary_parts]
    M = (2*mu*(inner(epsilon(u), epsilon(u)))+p)*dx
    tol = 1e-4
    PETScOptions.set('pc_type', 'asm')
    PETScOptions.set('sub_pc_type', 'lu')
    PETScOptions.set('pc_asm_overlap', '10')
    dw = TrialFunction(W)
    J = derivative(F, w, dw)
```

```
50    problem = NonlinearVariationalProblem(F, w, bcs, J)
51    solver_parameters = {"nonlinear_solver": "snes",
52    "snes_solver"      : { "linear_solver"    : "lu",
53      "absolute_tolerance": 1E-8,
54      "relative_tolerance": 1E-7,
55      "maximum_iterations": 20,
56      "report": True,
57      "error_on_nonconvergence": False
58        }}
59    if (ShouldRefine == True):
60      solver = AdaptiveNonlinearVariationalSolver(problem, M)
61      solver.parameters['nonlinear_variational_solver'].update(solver_parameters
            )
62      solver.solve(tol)
63      mesh = mesh.leaf_node()
64      (u, p) = w.leaf_node().split(True)
65    else:
66      solver  = NonlinearVariationalSolver(problem)
67      prm = solver.parameters
68      prm['newton_solver']['absolute_tolerance'] = 1E-8
69      prm['newton_solver']['relative_tolerance'] = 1E-7
70      prm['newton_solver']['maximum_iterations'] = 25
71      prm['newton_solver']['relaxation_parameter'] = 1.0
72      solver.solve()
73      (u, p) = w.split(True)
74    boundary_parts = MarkBoundaries(mesh,lx,ly)
75    return p, u, mesh, boundary_parts
76
77  def MarkBoundaries(mesh,lx,ly):
78  # Create mesh functions over the cell facets
79    boundary_parts = MeshFunction("size_t", mesh, mesh.topology().dim()-1)
80  # Mark all facets as sub domain 4
81    boundary_parts.set_all(4)
82
83  # Sub domain for noslip on obstacle
84    class NoslipOnObstacle(SubDomain):
85      def inside(self, x, on_boundary):
86        return on_boundary
87
88  # Mark obstacle noslip as sub domain 0
89    Gamma_s = NoslipOnObstacle()
90    Gamma_s.mark(boundary_parts, 0)
91
92  # Sub domain for Outflow (left)
93    class Outflow(SubDomain):
94      def inside(self, x, on_boundary):
95        return abs(x[0] - lx) < DOLFIN_EPS and on_boundary
96
97  # Mark outflow as sub domain 1
98    Gamma_o = Outflow()
99    Gamma_o.mark(boundary_parts, 1)
100
101 # Sub domain for Inflow (right)
102   class Inflow(SubDomain):
103     def inside(self, x, on_boundary):
```

```
104         return abs(x[0]) < DOLFIN_EPS and on_boundary
105
106 # Mark inflow as sub domain 2
107     Gamma_in = Inflow()
108     Gamma_in.mark(boundary_parts, 2)
109
110 # Sub domain for noslip on Top (Top)
111     class NoslipTop(SubDomain):
112       def inside(self, x, on_boundary):
113         return abs(x[1] - ly) < DOLFIN_EPS and on_boundary
114
115 # Mark top noslip as sub domain 3
116     Gamma_pos = NoslipTop()
117     Gamma_pos.mark(boundary_parts, 3)
118
119 # Sub domain for noslip on Bottom (Bottom)
120     class NoslipBottom(SubDomain):
121       def inside(self, x, on_boundary):
122         return abs(x[1]) < DOLFIN_EPS and on_boundary
123
124 # Mark bottom noslip as sub domain 3
125     Gamma_neg = NoslipBottom()
126     Gamma_neg.mark(boundary_parts, 3)
127     return boundary_parts
128
129 def AdjointSolver(mesh,boundary_parts,mu,u):
130     parameters['allow_extrapolation'] = True
131     V = VectorFunctionSpace(mesh, "Lagrange", 2)
132     Q = FunctionSpace(mesh, "CG", 1)
133     W = V * Q
134     phi = Constant((-1, 0))
135     noslip = Constant((0, 0))
136     bc0 = DirichletBC(W.sub(0), noslip, boundary_parts, 3)
137     bc1 = DirichletBC(W.sub(0), phi, boundary_parts, 0)
138     bc2 = DirichletBC(W.sub(0), noslip, boundary_parts, 2)
139     bcs = [bc0, bc1, bc2]
140     (v, q) = TestFunctions(W)
141     w = Function(W)
142     (z, r) = split(w)
143     f = Constant((0, 0))
144     F = (2*mu*(inner(epsilon(z), epsilon(v))) - div(v)*r +inner(dot(grad(u),v),z
          )+inner(dot(grad(v),u),z)- q*div(z)-inner(f, v))*dx
145     PETScOptions.set('pc_type', 'asm')
146     PETScOptions.set('sub_pc_type', 'lu')
147     PETScOptions.set('pc_asm_overlap', '10')
148     dw = TrialFunction(W)
149     J  = derivative(F, w, dw)
150     problem = NonlinearVariationalProblem(F, w, bcs, J)
151     solver  = NonlinearVariationalSolver(problem)
152     prm = solver.parameters
153     prm['newton_solver']['absolute_tolerance'] = 1E-8
154     prm['newton_solver']['relative_tolerance'] = 1E-7
155     prm['newton_solver']['maximum_iterations'] = 25
156     prm['newton_solver']['relaxation_parameter'] = 1.0
157     solver.solve()
```

```
158     (z, r) = w.split(True)
159     return r, z
160
161 def DragLagrangianDerivative(mesh,boundary_parts,mu,u,z,w,lx,ly,lam, Ao):
162     n = FacetNormal(mesh)
163     m1 = -2*mu*(inner(epsilon(u),epsilon(z)))*dot(w,n)*ds(0)
164     m2 = lam*dot(w,n)*ds(0)
165     Drag_derivative = assemble(m1+m2,exterior_facet_domains=boundary_parts)
166     return Drag_derivative
167
168 def ComputeDragLagrangian(mesh,boundary_parts,mu,p,u,lx,ly,lam,Ao):
169     n_infnty = Constant((1, 0))
170     n = FacetNormal(mesh)
171     I = Identity(2)
172     sigma = -p*I + 2*mu*epsilon(u)
173     M1 = -1.*dot(dot(sigma,n),n_infnty)*ds(0)
174     value1 = assemble(M1,exterior_facet_domains=boundary_parts)
175     g = assemble(Constant(1.0) * dx, mesh=mesh)-(lx*ly-Ao)
176     lagr_funct = value1+lam*g
177     return lagr_funct
178
179 def ArmijoLineSearch(mesh,boundary_parts,w,mu,U_o,lx,ly,lam,penalty_para,Ao):
180     oldmesh_coords = GetOldMeshCoordinates(mesh)
181     mesh_o = mesh
182     beta = 0.0001
183     alpha = 2.
184     if lam > penalty_para:
185       penalty_para = lam
186     p,u,mesh,boundary_parts = NSSolver(mesh,boundary_parts,lx,ly,mu,U_o,False)
187     r,z = AdjointSolver(mesh,boundary_parts,mu,u)
188     drag_lagran_o = PenaltyFunction(mesh,boundary_parts,mu,p,u,lx,ly,
            penalty_para,Ao)
189     der_drag_lagran = PenaltyFunctionDerivative(mesh,boundary_parts,mu,u,z,w,lx,
            ly,penalty_para,Ao)
190     mesh = UpdateMeshSolvingElasticityequation(mesh,boundary_parts,w,alpha)
191     p,u,mesh,boundary_parts = NSSolver(mesh,boundary_parts,lx,ly,mu,U_o,False)
192     drag_lagran = PenaltyFunction(mesh,boundary_parts,mu,p,u,lx,ly,penalty_para,
            Ao)
193     j = 1
194     while (drag_lagran > (drag_lagran_o+alpha*beta*der_drag_lagran) and alpha >
            1e-6):
195       mesh = ReStartOldMesh(mesh,oldmesh_coords)
196       alpha = 0.5*alpha
197       mesh = UpdateMeshSolvingElasticityequation(mesh,boundary_parts,w,alpha)
198       p,u,mesh,boundary_parts = NSSolver(mesh,boundary_parts,lx,ly,mu,U_o,False)
199       drag_lagran = PenaltyFunction(mesh,boundary_parts,mu,p,u,lx,ly,
              penalty_para,Ao)
200       j = j+1
201     return drag_lagran,alpha,penalty_para,mesh,boundary_parts
202
203 def CreateNewMeshUsingTriangle(mesh,lx,ly,boundary_parts):
204     num_cells = mesh.num_cells()
205     maxArea = 0
206     minArea = 1000
207     for i in cells(mesh):
```

```
208       if i.volume() > maxArea:
209         maxArea = i.volume()
210       if i.volume() < minArea:
211         minArea = i.volume()
212     boundary = BoundaryMesh(mesh,"exterior")
213     meshfile = open("obstaclemesh.poly", "w")
214     #Writing node information for the PSLG file
215     meshfile.write('%d \t 2 \t 0 \t 0 \n'%boundary.num_vertices())
216     coor = boundary.coordinates()
217     mapping = boundary.entity_map(0).array()
218     for i in vertices(boundary):
219       meshfile.write('%d \t %g \t %g \n'%(i.index(), coor[i.index()][0], coor[i.
              index()][1]))
220
221     #Writing edge information for the PSLG file
222     meshfile.write('%d \t 0 \n'%boundary.num_edges())
223     for i in edges(boundary):
224       meshfile.write('%d \t %d \t %d \n'%(i.index(),i.entities(0)[0],i.entities
              (0)[1]))
225
226 #  Since object translates, it is not centered at lx/2. and ly/2 aall the time
       . Therefore, we find the center using average of min/max of boundary x and
       y coordinates.
227     cordx = []
228     cordy = []
229     countnumofvertices = 0
230     for i in vertices(boundary):
231       if (coor[i.index()][0] != 0 and coor[i.index()][0] != lx and  coor[i.index
              ()][1] != 0 and coor[i.index()][1] != ly):
232         countnumofvertices = countnumofvertices+1
233         cordx.append(coor[i.index()][0])
234         cordy.append(coor[i.index()][1])
235     cx = (numpy.amax(cordx)+numpy.amin(cordx))/2.
236     cy = (numpy.amax(cordy)+numpy.amin(cordy))/2.
237
238     #Writing hole information for the PSLG file
239     meshfile.write('1 \n')
240     meshfile.write('1 \t %g \t %g'%(cx, cy))
241     meshfile.close()
242     triangle_syntax = './triangle'
243     triangle_syntax = triangle_syntax + ' -Qpqca'+str(maxArea) +' obstaclemesh'
244     os.system(triangle_syntax)
245
246     #  Read mesh and convert to dolfin format xml
247     os.system("dolfin-convert obstaclemesh.1.node shape.xml")
248     mesh = Mesh("shape.xml")
249
250     boundary_parts = MarkBoundaries(mesh,lx,ly)
251     return mesh, boundary_parts
252
253 def ReStartOldMesh(mesh,v):
254     V = VectorFunctionSpace(mesh, "Lagrange", 1)
255     u = Function(V)
256     coor = mesh.coordinates()
257     for i in vertices(mesh):
```

```
258       coor[i.index()][0] = v[i.index()][0]
259       coor[i.index()][1] = v[i.index()][1]
260     mesh.move(u)
261     return mesh
262
263 def UpdateMeshSolvingElasticityequation(mesh,boundary_parts,w, alpha):
264     V = VectorFunctionSpace(mesh, "Lagrange", 1)
265     u = TrialFunction(V)
266     v = TestFunction(V)
267     noslip = Constant((0, 0))
268     ww = Function(V)
269     ww_array = w.vector().array()
270     ww_array *= alpha
271     ww.vector()[:] = ww_array
272     bc0 = DirichletBC(V, noslip, boundary_parts, 3)
273     bc1 = DirichletBC(V, noslip, boundary_parts, 1)
274     bc2 = DirichletBC(V, noslip, boundary_parts, 2)
275     bc3 = DirichletBC(V, ww, boundary_parts, 0)
276     bcs = [bc0, bc1, bc2, bc3]
277     f = Constant((0, 0))
278     a = (inner(sigma(u),grad(v)))*dx
279     L = inner(f, v)*dx
280     u = Function(V)
281     solve(a == L, u, bcs)
282     mesh.move(u)
283     return mesh
284
285 def GetOldMeshCoordinates(mesh):
286     coords = mesh.coordinates()
287     n = mesh.num_vertices()
288     v = numpy.zeros((n,2))
289     for i in vertices(mesh):
290       v[i.index()][0] = coords[i.index()][0]
291       v[i.index()][1] = coords[i.index()][1]
292     return v
293
294 def SearchDirection(mesh,boundary_parts,mu,u,z,lx,ly, Ao):
295     parameters['allow_extrapolation'] = True
296     n = FacetNormal(mesh)
297     V = VectorFunctionSpace(mesh, "Lagrange", 1)
298     Q = FunctionSpace(mesh, "R", 0)
299     W = V * Q
300     ds = Measure("ds")[boundary_parts]
301     noslip = Constant((0, 0))
302     bc0 = DirichletBC(W.sub(0), noslip, boundary_parts, 3)
303     bc1 = DirichletBC(W.sub(0), noslip, boundary_parts, 1)
304     bc2 = DirichletBC(W.sub(0), noslip, boundary_parts, 2)
305     bcs = [bc0, bc1, bc2]
306     (w, p) = TrialFunctions(W)
307     (v, q) = TestFunctions(W)
308     g = assemble(Constant(1.0) * dx, mesh=mesh)-(lx*ly-Ao)
309     a = (inner(grad(w), grad(v))+inner(w,v))*dx+p*inner(v,n)*ds(0)+q*inner(w,n)*
          ds(0)
310     L = (2*mu*(inner(epsilon(u), epsilon(z))))*inner(n, v)*ds(0)-g*q*ds(0)
311     ww = Function(W)
```

```
312     solve(a == L, ww, bcs)
313     (w, p) = ww.split(True)
314     lam = p.vector().array().max()
315     return w,lam
316
317 def PenaltyFunction(mesh,boundary_parts,mu,p,u,lx,ly,lam,Ao):
318     n_infnty = Constant((1, 0))
319     n = FacetNormal(mesh)
320     I = Identity(2)
321     sigma = -p*I + 2*mu*epsilon(u)
322     M1 = -1.*dot(dot(sigma,n),n_infnty)*ds(0)
323     value1 = assemble(M1,exterior_facet_domains=boundary_parts)
324     g = assemble(Constant(1.0) * dx, mesh=mesh)-(lx*ly-Ao)
325     lagr_funct = value1+abs(lam*g)
326     return lagr_funct
327
328 def PenaltyFunctionDerivative(mesh,boundary_parts,mu,u,z,w,lx,ly,lam, Ao):
329     n = FacetNormal(mesh)
330     m1 = -2*mu*(inner(epsilon(u),epsilon(z)))*dot(w,n)*ds(0)
331     g = assemble(Constant(1.0) * dx, mesh=mesh)-(lx*ly-Ao)
332     Drag_derivative = assemble(m1,exterior_facet_domains=boundary_parts)
333     Drag_derivative = Drag_derivative - abs(lam*g)
334     return Drag_derivative
335
336 def WriteOutputDataFiles(u,p,z,r,w,k):
337     u_file = 'velocity.'+str(k)+'.pvd'
338     p_file = 'pressure.'+str(k)+'.pvd'
339     dir_file = 'direction.'+str(k)+'.pvd'
340     adjz_file = 'adjointz.'+str(k)+'.pvd'
341     adjr_file = 'adjointr.'+str(k)+'.pvd'
342     ufile_pvd = File(u_file)
343     ufile_pvd << u
344     pfile_pvd = File(p_file)
345     pfile_pvd << p
346     ufile_pvd = File(adjz_file)
347     ufile_pvd << z
348     pfile_pvd = File(adjr_file)
349     pfile_pvd << r
350     pfile_pvd = File(dir_file)
351     pfile_pvd << w
```

Vita

Chukwudi Chukwudozie was born in Abuja Nigeria to Patrick and Comfort Chukwudozie Okoli. He had his Pre-College education in Abuja and obtained a Bachelors in Chemical Engineering from the Federal University of Technology, Minna, Niger State Nigeria in January 2006. He obtained a Masters degree in Petroleum Engineering from Louisiana State University in 2011 and is currently a candidate for a Masters degree in Applied Mathematics, to be awarded in December 2015.

Permanent Address: No. 16B Obunagu Road
Awka, Anambra State, Nigeria
chuckdii2002@yahoo.com

This thesis was typeset with LaTeX by the author.