SENSITIVITY ANALYSES AND COMPUTATIONAL SHAPE OPTIMIZATION FOR INCOMPRESSIBLE FLOWS

John Burkardt

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

Doctor of Philosophy
in
Mathematics

May 1995
Blacksburg, Virginia

Keywords: Optimization, Fluid Flow

ABSTRACT

We consider the optimization of a cost functional defined for a fluid flowing through a channel. Parameters control the shape of an obstruction in the flow, and the strength of the inflow. The problem is discretized using finite elements. Optimization algorithms are considered which use either finite differences or sensitivities to estimate the gradient of the cost functional. Problems of scaling, local minimization, and cost functional regularization are considered. Methods of improving the efficiency of the algorithm are proposed.

ACKNOWLEDGEMENTS

I am grateful for the support of the Air Force Office of Scientific Research, through contracts 93-1-0061, 93-1-0280 and F49620-92-J-0261, which allowed me to carry out the research reported in this document.

I wish to thank the following persons:

- Avner Friedman of the IMA in Minneapolis, and Ken Bowers and John Lund of the Mathematics Department of the University of Montana, who sponsored conferences at which some of the ideas in this thesis were presented.
- My colleague Jeff Borggaard, for numerous informal discussions on the role and meaning of sensitivities, programming problems, scientific visualization, and the subtleties of $\LaTeX$ and $psfig$.
- Rossi Karamikhova, who shares the agony of preparing a dissertation, for her encouragement at several crucial moments.

Thanks are also due to my committee members, John Burns, Gene Cliff, and Terry Herdman, who welcomed me back into graduate school and, with various kindnesses, encouraged me to continue.

I express warm gratitude to Max Gunzburger, my adviser, who waited patiently as I progressed at my own, belabored speed, always prepared to consider any difficulties I had run into and offer help and insight.

I offer special thanks to Janet Peterson, who convinced me, despite my great misgivings, to return to graduate school and finish my degree. She went out of her way to ensure my admission, despite numerous lost documents; she guided my choice of classes; she urged me to begin writing as early as possible, and read several drafts; and she simply never doubted that I was going to finish.

Finally, I wish to thank my mother and father, who always believed in the value of learning, who were patiently supportive through my prolonged and discursive education, and who are possibly more excited about my degree than if they had gotten one themselves.

DEDICATION

My sister Barbara was born when I was 11, and she spent most of her early years riding on my shoulders. When our family moved across the country in a trailer, it was because of her that I got to sleep in the lowest, biggest bed (and it was over me that she had to crawl when she managed to fall out of that bed). When she got too big to carry on my shoulders, I used to swing her around, first by her arms, and then by her legs, and I only remember dropping her once. When I went to high school, she used to wait for me by the front door in the afternoon. When I went off to college, I left her on her own, but she turned out pretty well anyway. Now that she works at Hershey, she's taking care of me with regular shipments of chocolate.

In honor of our long happy years together, this thesis is dedicated to my dear sister Barbara.

Contents

List of Figures

List of Tables

Chapter 1

THE FOREBODY SIMULATOR PROBLEM

1.1 Introduction

In this chapter, we introduce an engineering problem that is an inspiration for our mathematical work. The *forebody simulator problem* seeks a simple obstacle that can be placed in a small wind tunnel so as to disturb the air flow in the same way that a given aircraft would.

Engineers can seek answers to this question by experiment and measurement. But new and general insights may be found by applying mathematical analysis to this problem. For a mathematician, the first step in studying a physical problem involves creating a *model*, that is, an abstract version of the problem that is simple enough to analyze, but which retains the important features of the original.

The modeling process begins with questions about how the *geometry* of the physical problem will be simplified and represented in the abstract problem. For the forebody simulator problem, we will replace the complicated three-dimensional geometry of the wind tunnel by a simple two-dimensional channel. The obstacle to be placed in the flow will be represented by a mathematical curve, which itself will be describable by just a few numbers. After we have specified the geometry of our abstract problem, we will be ready for the question of how to determine the *behavior* of the fluid as it moves inside the region.

1.2 The Physical Problem: A Forebody Simulator

A motivating problem for this work comes from the field of aircraft engine development. Before attaching an experimental engine to an aircraft and testing it in flight, it is vital to gain some idea of how the engine will perform, by placing it in a wind tunnel and subjecting it to a variety of wind conditions. Wind tunnels cannot be built large enough to hold an entire aircraft, but for studies of engine performance, it is enough if the wind tunnel can hold the "forebody", that is, the portion of the aircraft up to and including the engine. However, there are aircraft so large that even the forebody cannot fit into a wind tunnel.

Out of this necessity, engineers have designed *forebody simulators*, which are typically short, stubby shapes that fit within the wind tunnel and roughly reproduce the flow disturbances of the larger forebody. The design of these simulators is generally done by trial and error. A mockup is constructed out of plywood or steel, attached to the engine and placed in the wind tunnel, as shown in Figure 1.1. Measurements of the flow going into the engine are compared with rough data for the true forebody. If these measurements are found to be close enough, then the wind tunnel tests of the engine may proceed. Otherwise, a new mockup is designed, with modifications made according to the intuition of the engineers as to what would reduce the discrepancies with the desired flow profile.

Naturally, this method of designing a forebody simulator is crude, tedious, and expensive. Engine designers want to build and test their preliminary forebody simulators on a computer. However, while there is much experience with computing the behavior of air flow in a given, fixed region, it is a much more difficult matter to attempt to analyze the flow over *every*

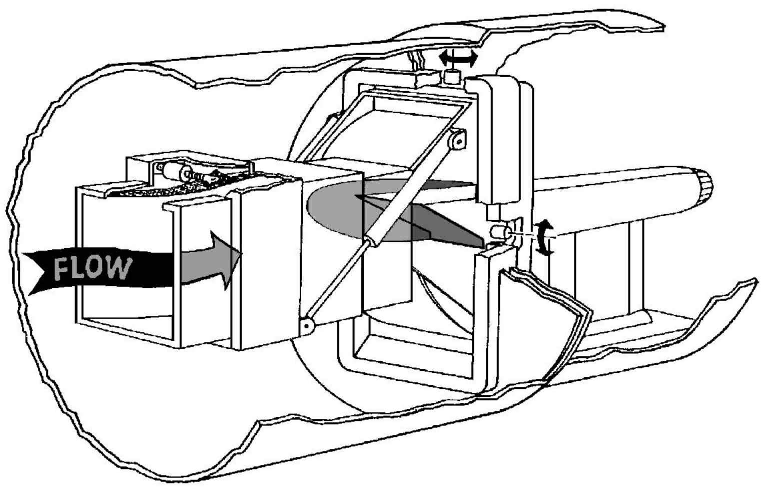

Figure 1.1: An engine and forebody simulator, in a wind tunnel.

possible region, in order to find the design with the best properties.

We will be particularly interested in considering the difficulties and problems that can occur in the various stages of modeling this problem, discretizing it, and constructing a suitable computational algorithm for its solution.

A complete description of the forebody simulator problem is available in Huddleston [18] and in Borggaard, Burns, Cliff and Gunzburger [2].

1.3 An Abstract Model of the Forebody Simulator

We don't want to try to solve the engineer's original problem. We are interested in making a much simplified version that can be formulated mathematically, and solved computationally. We hope that we retain enough of the "interesting" features so that our results are still applicable.

We simplify the geometry of the problem by moving to two dimensions. We replace the complicated structure of the wind tunnel by a simple channel with parallel sides. We assume that there is a small bump or obstruction placed somewhere along the bottom of the channel. The shape and size of this bump will be determined by the values of a few numbers, which we will call "bump parameters". In particular, if the bump parameters are all zero, there will be no bump.

As for the fluid flowing in this channel, we will assume that it is incompressible, has a nonzero viscosity, and flows in a steady fashion.

We must now consider the boundary conditions for this flow, that is, its behavior along the walls and openings of the flow region. The top and bottom of the channel will be impermeable walls. Immediately next to walls, there is no fluid motion at all. We will assume that the fluid enters the region on the left hand side, and that the fluid velocities along this boundary are completely specified by some "inflow function". This inflow function, in turn, should be describable by the values of a few "inflow parameters". Again, we assume the convention that if all the inflow parameters are zero, no fluid enters the region. The right side of the region, where the flow exits, is called the "outflow" side. For mathematical reasons, we will need to assume that the flow has "settled down" at this point, which will allow us to specify that the vertical velocity is zero, and the horizontal velocity does not change along the horizontal direction. The simplified model of our problem is shown in Figure 1.2.

We haven't specified the number of bump and inflow parameters, their values, or the exact nature of their influence on the bump shape and inflow function. In fact, we intend to leave these quantities unspecified for now. Our original problem required us to find the shape, from among *all possible* shapes, which had the best behavior. In our framework, instead of considering all possible shapes, we restrict ourselves to some particular *space* or family of parameterizable curves to represent our bump. Any curve which is an element of this space may then be specified by giving the values for the bump parameters.

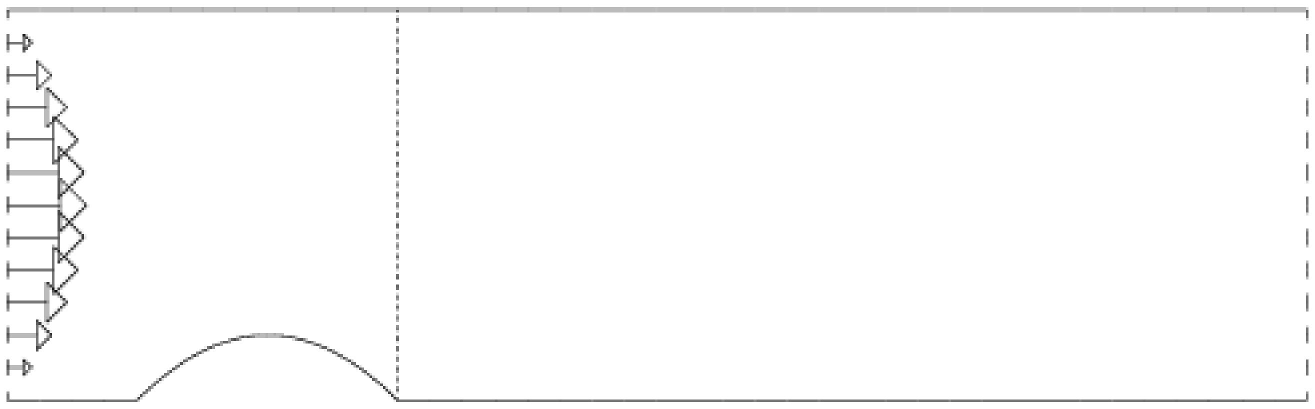

Figure 1.2: An abstract model of the forebody simulator problem.

Once we have chosen values for the bump and inflow parameters (and for the Reynolds number, to be discussed in the next chapter), the flow problem is fully described. As long as the inflow velocities are not too large relative to the size of the region and the fluid viscosity, there will be exactly one corresponding flow solution, that is, just one way that the fluid will move through the region.

This solution can be represented by horizontal and vertical velocities u and v, and a pressure p. These quantities are called the “state variables”. For our problem, we expect these quantities to vary continuously, and in fact, differentiably with position throughout the flow region. Because we have specified steady flow, however, the flow quantities will *not* depend on time.

In the interior of the region, usually right behind the bump, we will place a “profile line”, along which we will assume that measurements of all the flow quantities can be made. This profile line corresponds to the engine inlet in the original free jet problem. We may assume that we have been given a set of ideal or “target” values for the flow variables along this line, and that we are seeking a flow whose measurements along the profile line reproduce, or approximate, those target measurements.

The total discrepancy between the target measurements and the measurements corresponding to any particular bump will be summarized in a single number which is called the "cost functional". In particular, if we are able to achieve a perfect match, the corresponding cost would be zero. We can thus describe our effort more abstractly, as seeking to minimize the cost functional, given that we are allowed to vary the parameters.

Our method of producing the flow which best matches the given profile will be contained in a separate *optimization algorithm.* We will give this algorithm an initial guess for the best set of parameters, and it will give us a new set of parameters to investigate. The algorithm then expects us to evaluate the cost for the fluid flow that corresponds to this set of parameters. The optimization algorithm will usually also need derivative information about the cost functional, that is, information about the partial derivatives of the cost with respect to the parameters. We will see that producing such information accurately and efficiently is a difficult task.

In order to evaluate the cost, of course, we must solve the fluid flow problem determined by the given parameters. It should be clear, then, that we will need to be able to solve many fluid flow problems. In fact, we will find that the underlying fluid flow solutions are the overwhelming expense of our algorithm.

We shall now consider the form of the equations involved in the fluid flow problem, and practical methods of representing and solving such a problem.

Chapter 2

A PARAMETERIZED CLASS OF FLOW PROBLEMS

2.1 Introduction

In our research, we will look at a range of flow problems. These problems will have a great deal in common, and will differ only in a few small ways. In this chapter, we will define precisely how we generate the different problems, and how we can specify a few numbers that will completely describe any particular flow problem we are interested in.

This specification will consist of giving the number and values of a set of *flow parameters*, which will in turn control the Reynolds number, the inflow strength, and the shape of an obstructing bump. After describing how each of these parameters enters the problem, we finish by displaying a sample problem.

2.2 How the Parameters Enter the Problem

We begin by specifying the parts of our flow problem that will *never* change. Our flow region, which we will symbolize by Ω, will be a rectangular channel, with opposite corners at $(0, 0)$ and $(xmax, ymax)$. A bump of an unspecified shape lies along the bottom of this channel, extending from $(1, 0)$ to $(3, 0)$. Over this range, the height of the bump is given by $y = Bump(x, \alpha)$, where the *Bump* function has a simple form that depends on one or more parameters that we call α. We prefer to think that the *Bump* function is always nonnegative, but this is not essential to our work. In fact, it will be more convenient to allow *Bump* to take on negative values as well. In such a case, negative values of α would mean that our region would actually be enlarged, as the bottom of the channel is "hollowed out".

The boundary of our flow region, which we may symbolize by Γ, is simply the top, bottom, left and right extremities. The bottom extremity is not rectilinear, since it includes the surface of the bump. The left extremity may be called the inflow boundary, and the right extremity the outflow boundary.

The boundary conditions on u and v, the horizontal and vertical components of the velocity function, and on p, the pressure, will have the general form:

- $v(x, y) = 0$ on the boundary;
- $u(0, y) = Inflow(y, \lambda)$ where the $Inflow$ function depends on one or more parameters λ, as discussed in Section 2.4;
- $u(x, y) = 0$ on the upper and lower walls and the surface of the bump;
- $\dfrac{\partial u}{\partial n}(xmax, y) = 0$;
- $p(xmax, ymax) = 0$.

Finally, the fluid flowing in the channel has a positive kinematic viscosity ν which we have not yet specified. For convenience, instead of specifying ν, we will specify *Re* which we take to be $1/\nu$, and which, if the equations are appropriately nondimensionalized, we may call the Reynolds number. Specifying a particular positive value of *Re* is then equivalent to specifying ν.

We should note that a more common definition for the Reynolds number, at least among experimentalists, involves the ratio:

$$Re \equiv \frac{\rho v l}{\mu} \tag{2.1}$$

where ρ is a typical density, v a typical velocity, l a typical length, and μ the absolute viscosity. Our kinematic viscosity ν is equal to $\frac{\mu}{\rho}$. Since the velocity does not appear in our version of the Reynolds number, if we wish to translate our results in terms of the experimentalist Reynolds number, we must assume that a velocity on the order of 1 is a "typical velocity".

If we agree that all our flow problems will have these characteristics, then to completely specify a particular problem, we simply need to specify the value of Re, the number of parameters λ and α that we will use, the values of those parameters, and their meanings, that is, how the parameters are used to determine the $Bump$ and $Inflow$ functions.

Typically, when we wish to define a particular class of problems, we will specify a form for the $Bump$ and $Inflow$ functions, and the number of parameters to use for each; but we are not yet ready to specify fixed numerical values of λ, α and Re.

For the moment, we will assume that our choice of values for these parameters is *unconstrained*; that is, any set of parameter values is worth considering, in the sense that it corresponds to a physically meaningful problem. Generally, this sweeping assumption is not strictly true. We won't want negative Reynolds numbers, or bumps that protrude through the top of the region, or inflow functions that actually flow out! However, we will proceed as though these undesirable situations are unlikely to crop up in any of our problems.

As long as a choice of parameter values represents a legitimate problem, that is, a problem for which a physically sensible flow solution can be computed, then we call the values a *feasible set of parameter values* (meaning that it results in a meaningful flow) and we call the corresponding flow solution a *feasible flow solution* (meaning that it is the result of a particular choice of parameter values).

By using parameters, we have (partially) discretized the specification of the problem. Instead of considering absolutely arbitrary bump and inflow shapes, we only consider various numbers as possible values for the bump and inflow parameters. Instead of ranging over function spaces, we range over R^n. This makes it easy to define a particular problem, and search among a family of related problems.

We will now briefly discuss the meaning of the three different sets of parameters.

2.3 The Reynolds Number

The Reynolds number *Re* will appear in the Navier Stokes equations, which determine the fluid flow in our region. These equations will not be discussed until Chapter 3, but we wish to comment here on the role that the Reynolds number plays as a parameter.

The Navier Stokes equations involve a balance between "diffusion" and "convection" terms. The diffusion terms are linear and, by themselves, would tend to produce a simple flow. The convection terms are nonlinear, harder to solve, and introduce complications into the flow. The Reynolds number multiplies the convection terms, and hence controls the relative strength of this disruptive nonlinear influence.

In a diffusion-dominated flow, disturbances tend to die out, dampened by the fluid viscosity. Imagine, for instance, how the behavior of ripples in a pond would change if we replaced the water by cold maple syrup. Such a "thick" fluid tends to move in a very smooth and stable way, changing only slightly in response to small changes in local conditions, the boundary conditions, or the shape of the region.

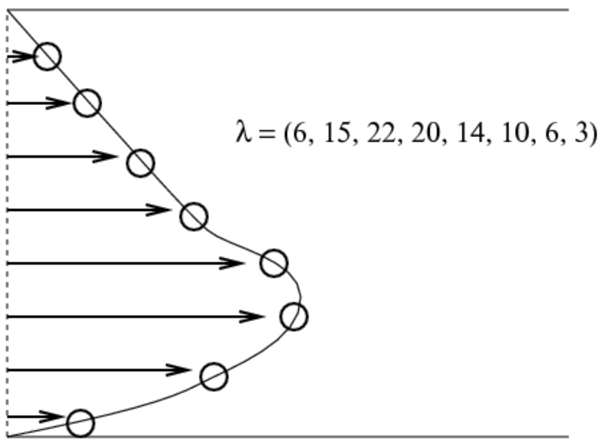

Figure 2.1: An inflow specified with eight parameters.

As Re increases, the convection terms dominate the flow. Now the flow field may change greatly over a small spatial region. The flow field solution is much more sensitive to changes in the problem conditions. Because of these facts, high Reynolds number flow problems are intrinsically harder to solve accurately.

We will not be solving the Navier Stokes equations directly, but the discretized version that we will end up working with inherits many of these characteristics. In particular, we will find that solutions to the discretized problem get very hard to compute for large values of Re. In some cases, we will only be able to find such a solution by solving a series of problems at lower values of Re, building up to a solution at the desired value of Re.

2.4 Specifying the Inflow Function

The next item to consider is the $Inflow$ function, which specifies the boundary condition to be applied to the horizontal velocity u along the left opening of the channel. While, in a real situation, there might be almost any sort of inflow data, we will want to set up a scheme where just a few numbers, represented by λ, are enough to specify the value of the inflow

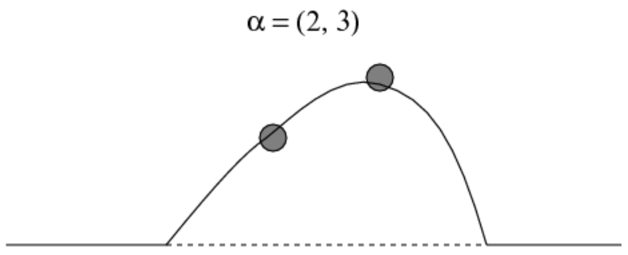

Figure 2.2: A bump specified with two parameters.

data at every point along the opening.

A natural way to do this is to let the values of λ represent the value of the inflow data at regularly spaced points along the opening, and then “fill in” the behavior of the function at other points in some simple way. While a wide variety of polynomials or piecewise polynomials may be used for this purpose, we will generally use *cubic splines* [9].

In Figure 2.1 we give an example of an inflow function which has been constructed by specifying eight desired values, which are listed in the figure. The intermediate behavior of the function is entirely determined by this data, and by the condition that the function be zero at both endpoints.

2.5 Specifying the Bump Shape

It remains for us to show how the height of the bump is to be described, that is, the form of the function $y = Bump(x, \alpha)$. The fact that we require the bump height y to be an explicit function of x means we have ruled out bumps which “fold over” or otherwise violate the requirement that the bump height be parameterizable in terms of x.

We represent the bump height by setting it to zero at its endpoints, and specifying its value at regularly spaced points in between. We symbolize this set of values by α. We then construct the appropriate interpolating function so that we can produce values of the

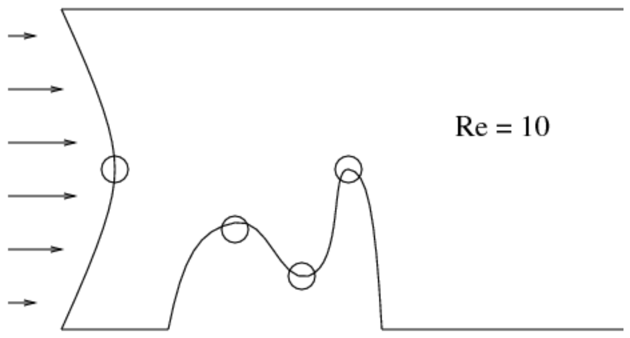

Figure 2.3: The flow problem $(\lambda, \alpha_1, \alpha_2, \alpha_3, Re) = (1.0, 1.0, 0.5, 1.5, 10.0)$.
The flow region actually extends further to the right.

bump at any point over the interval defined by its endpoints. Usually we will take this interpolating function to be a cubic spline, although we may also choose piecewise linear or piecewise quadratic functions. In Figure 2.2 we show the generation of a typical bump shape, requiring the specification of two parameters, where interpolation by a cubic spline was used.

2.6 Example: A Five Parameter Problem

Now that we have detailed how any particular problem can be specified by giving the value of Re, and the number and values of the parameters α and λ, we will show a specific example.

The inflow boundary condition will be described by a single value $\lambda = 1.0$, the bump by the three values, $\alpha = (1.0, 0.5, 1.5)$, and the Reynolds number will be set to 10, so that the problem data can be summarized as:

$$(\lambda, \alpha_1, \alpha_2, \alpha_3, Re) = (1.0, 1.0, 0.5, 1.5, 10.0). \tag{2.2}$$

A schematic diagram of the corresponding problem is displayed in Figure 2.3.

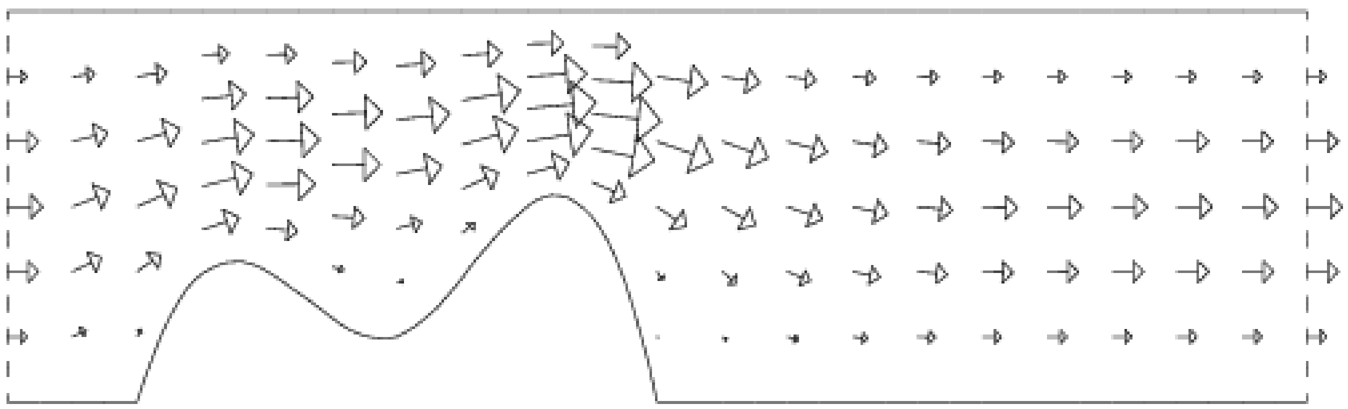
Figure 2.4: The velocity solution for the five parameter problem.

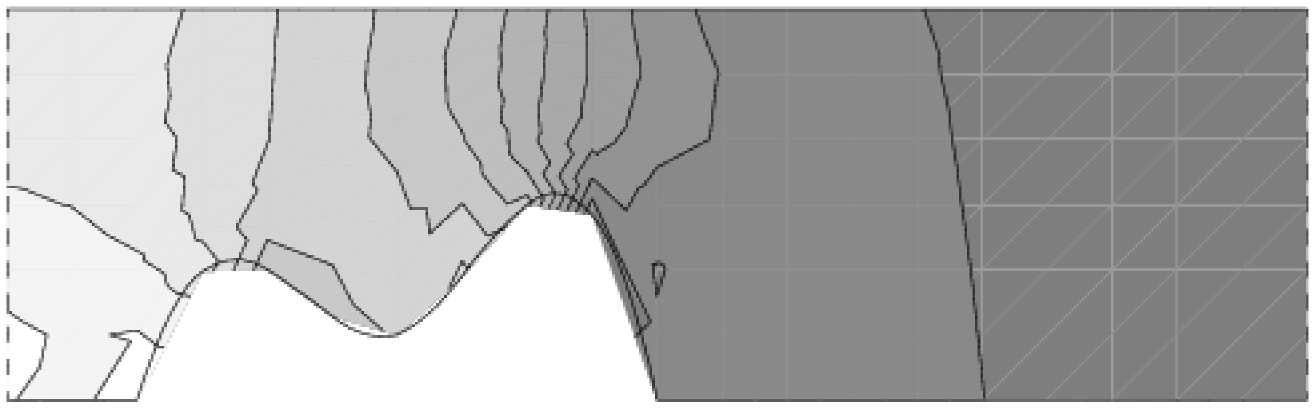
Figure 2.5: Contours of the pressure solution for the five parameter problem.

Once we have defined the problem, the Navier Stokes equations, which we discuss shortly, will determine the behavior of the fluid at any point within the flow region. To emphasize this point, Figures 2.4 and 2.5 display the velocity and pressure solution fields of an approximate solution of the Navier Stokes equations, for the data we have just discussed. From the plots, the influence of the bump on the flow should be quite clear.

Now that we understand how the parameters will define the problem, it is time to turn to the question of how the problem can be solved, that is, how we can come up with values of the velocity and pressure that satisfy the boundary conditions and the physical laws that govern fluid behavior. To begin, we will look at the differential equations that describe the behavior of an incompressible viscous fluid.

Chapter 3

CONTINUOUS FLUID FLOW EQUATIONS

3.1 Introduction

In this chapter, we describe the *Navier Stokes equations.* These equations constrain, and generally determine, the behavior of the fluid flow in a given problem. In particular, they apply to the parameterized class of flow problems discussed in Chapter 2.

We examine the solution of a special example of our flow problems, called *Poiseuille flow*, which is useful in checking the correctness of the computational algorithms, and in understanding the behavior of flow solutions to problems with no bump, or with relatively low inflow velocity conditions.

We note that most flow problems are very difficult to solve. We then discuss *Newton's method*, which shows us how to turn a hard, nonlinear problem, into an easier sequence of linear problems.

3.2 The Navier Stokes Equations

The steady incompressible flow of a viscous fluid in a two dimensional region may be completely described by three functions of position: the horizontal velocity $u(x, y)$, vertical velocity $v(x, y)$, and pressure $p(x, y)$. The quantities u and v are really vector components of a single physical velocity which we may alternately write as $\mathbf{u}$.

We may refer to these functions as the *flow field solution*. They are an example of a set of *state variables*, that is, information which completely describes the state of some physical system.

Because they represent the behavior of a physical fluid, the functions u, v, and p obey certain physical laws. Given our assumptions about the problem, we will find it appropriate to assume that these flow functions satisfy the Navier Stokes equations for stationary incompressible viscous flow at every point (x, y) within the flow region Ω.

A compact vector form of these equations may be written as:

$$\begin{aligned} -\Delta \mathbf{u} + Re(\mathbf{u} \cdot \nabla \mathbf{u} + \nabla p) &= 0 && (3.1) \\ \nabla \cdot \mathbf{u} &= 0 && (3.2) \end{aligned}$$

while we will prefer the verbose, but more readily grasped scalar version of the equations:

$$\begin{aligned} -(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}) + Re(u\frac{\partial u}{\partial x} + v\frac{\partial u}{\partial y} + \frac{\partial p}{\partial x}) &= 0 && (3.3) \\ -(\frac{\partial^2 v}{\partial x^2} + \frac{\partial^2 v}{\partial y^2}) + Re(u\frac{\partial v}{\partial x} + v\frac{\partial v}{\partial y} + \frac{\partial p}{\partial y}) &= 0 && (3.4) \\ \frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} &= 0 && (3.5) \end{aligned}$$

Because they constrain the behavior of state variables, these equations may be referred to as the *state equations*. The first pair of equations are sometimes referred to as the *equations* *of motion* or the *horizontal* and *vertical momentum equations.* The last equation is the *continuity* or *incompressibility equation.* The quantity Re is the Reynolds number.

While the Navier Stokes equations constrain the behavior of the fluid within the flow region, a flow problem is only completely described when the Reynolds number and certain information about the flow along the boundary is given as well. In Chapter 2 we discussed the sort of boundary information that would be supplied for our problems, and we will assume that the boundary information and the value of Re necessary to complete the problem have all been specified.

We have said that the Navier Stokes equations constrain any flow solution. That is, *if* there is a solution to the given problem, then it must satisfy the Navier Stokes equations. However, we are interested in knowing more about the behavior of possible solutions. Our questions include existence, uniqueness, and continuous dependence on data, or more precisely:

- Is there always at least one solution to a given problem?
- Is the solution to a given problem unique?
- Does the solution depend continuously, or even differentiably, on the boundary data and the Reynolds number?

For arbitrary boundary data, we cannot guarantee anything. Only if the boundary data satisfy certain "consistency" requirements, and if the region itself is bounded with a "reasonably smooth" boundary, can we guarantee the existence of at least one flow solution for any positive Reynolds number. It is easy to see that some constraints must be placed on boundary conditions. Otherwise, for instance, we could specify that fluid enter the region from all sides, an impossibility for incompressible flow.

If, furthermore, the Reynolds number is sufficiently small, then we can also guarantee that there is just one such flow solution.

Once the Reynolds number reaches a certain critical value, then uniqueness can no longer be guaranteed. If we consider the graph of the flow solution versus Reynolds number, then the graph may split or bifurcate into two or more distinct solution branches once a critical Reynolds value is reached. Such splitting can recur at certain higher Reynolds numbers as well.

However, below the first critical Reynolds value, and on any single solution branch, the flow solution will be continuously differentiable in terms of the Reynolds number.

For a summary of what is known about existence, uniqueness, and continuous dependence on the Reynolds number and boundary conditions, see Ladyzhenskaya [21] or Temam [25].

We will not delve further into these questions of existence and uniqueness. Instead, we will assume that when we pose one of our flow problems, there will be, at least locally, exactly one flow solution, which will depend smoothly on each of the problem parameters. We still face the daunting problem of *producing* that solution. To get an explicit formula for a solution to one of our problems requires the solution of a set of nonlinear partial differential equations. Even for the simple problems we will consider, there is no known method of producing such a solution. When faced with a general problem, then, we will find that we must turn to approximate methods in order to produce any useful information about a solution.

Before considering such methods, we turn to a special problem, for which an exact solution of the Navier Stokes equations is known.

3.3 Poiseuille Flow

Consider a horizontal channel (with no obstruction) of constant height *ymax*, and length *xmax*. Suppose that an incompressible, viscous fluid moves through the channel in accordance with the steady Navier Stokes equations. Suppose that the boundary conditions imposed on the flow are:

$$\begin{aligned}
v(x,y) &= 0 \text{ along the boundary;}\\
u(0,y) &= \lambda y(ymax - y) \text{ along the inflow boundary;}\\
u(x,y) &= 0 \text{ along the upper and lower walls;}\\
\frac{\partial u}{\partial n}(xmax,y) &= 0 \text{ along the outflow boundary;}\\
p(xmax,ymax) &= 0.
\end{aligned}$$

Then the Navier Stokes equations and boundary conditions are exactly satisfied by the following set of state variable functions:

$$u(x,y) = \lambda y(ymax - y) \tag{3.6}$$

$$v(x,y) = 0 \tag{3.7}$$

$$p(x,y) = 2\lambda(xmax - x)/Re. \tag{3.8}$$

This exact solution can be used as a very basic test of the computational correctness of our approximate Navier Stokes solver. The solution also makes it easy to compute the explicit form of the associated cost function, and gradients of the cost function with respect to the parameters λ and *Re*, and to compare them to the results arrived at by computation. Finally, at least for small values of *Re* and the inflow, we can introduce a small bump into the channel and still have the flow tend to return to the Poiseuille solution a short distance downstream from the bump.

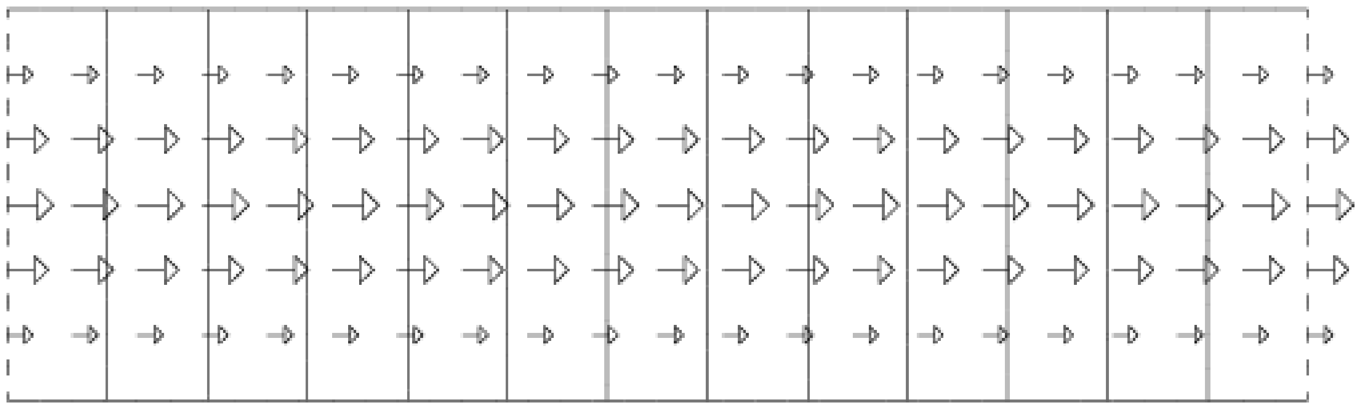

Figure 3.1: Velocity vectors and pressure contour lines for Poiseuille flow.

In some ways, the Poiseuille solution is not an ideal example. It's much too simple. It is very carefully constructed so that the nonlinear terms in the Navier Stokes equation exactly cancel out. This means that we will not see some of the nonlinear behavior that makes a general fluid flow problem so hard to solve.

3.4 Picard Iteration for the Continuous Navier Stokes Equations

Since there are no practical direct methods of solving the Navier Stokes equations over an arbitrary region, we need to consider methods of computing approximate solutions. While eventually we will turn to methods of discretization, we first consider several *iterative* schemes, which work directly with the original, continuous problem.

The primary appeal of these iterative schemes is that they replace a hard problem (solving a system of nonlinear partial differential equations) by a sequence of somewhat easier problems (solving a system of linear partial differential equations). While it is still beyond our abilities to solve even the linearized system, the ideas we discuss will help us to see how to handle similar problems when we finally move to the discretized case.

The first method we will discuss is known as *simple iteration* or *Picard iteration.* The idea is that we can try to solve the problem

$$F(X) = 0, \tag{3.9}$$

by "splitting" F into a form such as:

$$F(X) = X - G(X), \tag{3.10}$$

or, in general, finding a "splitting" function $H(X, Y)$ so that:

$$F(X) = H(X, X). \tag{3.11}$$

Picard iteration would then consist of picking a starting point X_0 and solving the sequence of (generally implicit) problems:

$$H(X_{n+1}, X_n) = 0. \tag{3.12}$$

In many cases, if the decomposition that defines a Picard iteration is chosen properly, the iteration will converge for a wide range of starting guesses X_0, although convergence might be quite slow.

To apply Picard iteration to our flow problem, let us assume that we have a starting estimate (u_0, v_0, p_0) of the solution, which can be any functions of (x, y) defined over Ω, and which need not satisfy the boundary conditions.

We now rewrite the Navier Stokes equations and the boundary conditions so that all terms appear on the left hand side only, so that formally we have reproduced the generic nonlinear system $F(X) = 0$. For our problem, this would result in the system of equations:

$$-(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}) + Re(u\frac{\partial u}{\partial x} + v\frac{\partial u}{\partial y} + \frac{\partial p}{\partial x}) \quad = \quad 0 \tag{3.13}$$

$$
\begin{aligned}
-(\frac{\partial^2 v}{\partial x^2} + \frac{\partial^2 v}{\partial y^2}) + Re(u\frac{\partial v}{\partial x} + v\frac{\partial v}{\partial y} + \frac{\partial p}{\partial y}) &= 0 && (3.14)\\
\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} &= 0 && (3.15)\\
v(x_\Gamma, y_\Gamma) &= 0 && (3.16)\\
u(0,y) - Inflow(y,\lambda) &= 0 && (3.17)\\
u(x_{wall}, y_{wall}) &= 0 && (3.18)\\
\frac{\partial u}{\partial n}(xmax, y) &= 0 && (3.19)\\
p(xmax, ymax) &= 0 && (3.20)
\end{aligned}
$$

Here, the equation

$$v(x_\Gamma, y_\Gamma) = 0 \qquad (3.21)$$

is meant to assert that the vertical velocity v is zero at every point (x, y) that lies on the flow region boundary Γ. A similar symbolism is used to assert the conditions that apply just on the upper and lower walls.

Our decomposition of $F(X)$ into $H(X_{n+1}, X_n)$ will simply involve replacing each nonlinear term by a term that is linear in the "new" variables. Rather than subscripting our state variables, we will use the symbols u_{old}, v_{old} and p_{old} to refer to the data at the old point. Then our equations become:

$$
\begin{aligned}
-(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}) + Re(u_{old}\frac{\partial u}{\partial x} + v_{old}\frac{\partial u}{\partial y} + \frac{\partial p}{\partial x}) &= 0 && (3.22)\\
-(\frac{\partial^2 v}{\partial x^2} + \frac{\partial^2 v}{\partial y^2}) + Re(u_{old}\frac{\partial v}{\partial x} + v_{old}\frac{\partial v}{\partial y} + \frac{\partial p}{\partial y}) &= 0 && (3.23)\\
\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} &= 0 && (3.24)\\
v(x_\Gamma, y_\Gamma) &= 0 && (3.25)\\
u(0,y) - Inflow(y,\lambda) &= 0 && (3.26)\\
u(x_{wall}, y_{wall}) &= 0 && (3.27)
\end{aligned}
$$

$$\begin{aligned} \frac{\partial u}{\partial n}(xmax, y) &= 0 \qquad (3.28) \\ p(xmax, ymax) &= 0 \qquad (3.29) \end{aligned}$$

By linearizing four terms in our equations, we have come up with a linear system of partial differential equations, which are presumably easier to solve repeatedly than our original nonlinear system.

We will now discuss Newton's method for solving our nonlinear system. We will see that Newton's method converges faster, but only if the initial guess is close enough to the correct answer. This will mean that our best method will be a hybrid of Picard iteration, to move from an arbitrary starting point close to the correct solution, and Newton's method, which will rapidly finish the iteration once we are near enough.

3.5 Newton's Method for the Continuous Navier Stokes Equations

Newton's method for solving a system of nonlinear equations $F(X) = 0$ assumes that we have an approximate solution X_0 and a linearized operator $FP(X)$ which allows us to approximate the behavior of the function F around any point X as:

$$F(X) = F(X_0) + FP(X_0)\ (X - X_0) + O((X - X_0)^2). \qquad (3.30)$$

If $F(X)$ is a single algebraic equation, then $FP(X)$ is simply the usual derivative function. If $F(X)$ is a system of algebraic equations, then $FP(X)$ is the Jacobian matrix. For our problem, however, $F(X)$ is not a function of a few real numbers, but of several real valued functions. The computation and use of the linearization operator FP will be more complicated than in the discrete case, but the idea will still be the same.

Now if we drop the error term in Equation (3.30) and replace $F(X)$ by 0, we may solve for

X, and arrive at the Newton estimate for the root:

$$X_1 = X_0 - FP^{-1}(X_0)\ F(X_0). \tag{3.31}$$

If our initial approximation is close to the correct answer X^*, and if the differential operator FP is continuous and invertible at X^*, then we may expect that repeated application of this operation will produce a sequence of points $\{X_k\}$ which converge to X^*. We defer a detailed consideration of the convergence of Newton's method to the next chapter, when we discuss the discrete case that we will actually need to solve.

To apply Newton's method to our flow problem, let us assume that we have a starting estimate (u_0, v_0, p_0) of the solution, which can be any functions of (x, y) defined over Ω, and which need not satisfy the boundary conditions. This estimate might, in fact, be the output of the Picard iteration.

We now rewrite the Navier Stokes equations and the boundary conditions so that all terms appear on the left hand side only, as in Equations (3.13) through (3.20), so that formally we have reproduced the generic Newton system $F(X) = 0$.

We may then symbolize the left hand side of these equations as $F(u, v, p)$ and our search for a solution of the original problem can be represented as seeking functions (u, v, p) so that:

$$F(u, v, p) = 0. \tag{3.32}$$

Now we need to compute the linearization operator FP associated with our nonlinear function. If our variables were simple algebraic quantities, then we would simply take the usual partial derivative $\frac{\partial}{\partial u}$. A term like u^2 would result in a derivative term of $2u$. However, our variables are functions and it is not immediately clear how to differentiate equations involving spatial derivatives of these functions, such as $u\frac{\partial u}{\partial x}$. A naive guess for the partial derivative might be the meaningless result $*\frac{\partial u}{\partial x} + u\frac{\partial *}{\partial x}$.

What's wrong with this result does suggest the proper approach, however. Where we placed a "*" in the above expression, we actually need something to operate on. Suppose we try to compute something like the *differential* instead of the derivative of our function? For an algebraic system, we would merely be computing a result like $2u\ du$, where the du seems to add little. But for our situation, the equivalent of the du will supply the missing operand.

Since the arguments of our functions aren't scalars, but are themselves functions, we must proceed carefully. In order to use a difference quotient approach, we will need to have a family of perturbations parameterized by a scalar ϵ. So we first consider some arbitrary set of perturbation functions, which we will call $\tilde{u}$, $\tilde{v}$ and $\tilde{p}$. We will then evaluate the function F at the family of "nearby" points $(u_0 + \epsilon\tilde{u}, v_0 + \epsilon\tilde{v}, p_0 + \epsilon\tilde{p})$ and use a difference quotient to analyze the behavior of F as $\epsilon \to 0$.

Our difference quotient will have the form:

$$FP(u_0, v_0, p_0)(\tilde{u}, \tilde{v}, \tilde{p}) = \lim_{\epsilon \to 0} \frac{F(u_0 + \epsilon\tilde{u}, v_0 + \epsilon\tilde{v}, p_0 + \epsilon\tilde{p}) - F(u_0, v_0, p_0)}{\epsilon}. \tag{3.33}$$

If this limit exists, we will have found what is called the *Gateaux derivative* of F, which is a sort of directional derivative in abstract spaces. The derivation of the Gateaux derivative requires that we pick a particular direction or perturbation. If in fact the result is independent of the direction chosen, then we have actually found the *Frechet derivative.* The defining property of the Frechet derivative FP(x)(h) is that

$$\lim_{\|h\| \to 0} \frac{\|F(x+h) - F(x) - FP(x)(h)\|}{\|h\|} = 0. \tag{3.34}$$

If we carry out the limit operation, the resulting operator $FP(u_0, v_0, p_0)(\tilde{u}, \tilde{v}, \tilde{p})$ has the following form:

$$-(\frac{\partial^2 \tilde{u}}{\partial x^2} + \frac{\partial^2 \tilde{u}}{\partial y^2}) + Re\ (\tilde{u}\frac{\partial u_0}{\partial x} + u_0\frac{\partial \tilde{u}}{\partial x} + \tilde{v}\frac{\partial u_0}{\partial y} + v_0\frac{\partial \tilde{u}}{\partial y} + \frac{\partial \tilde{p}}{\partial x}) \tag{3.35}$$

$$-(\frac{\partial^2 \tilde{v}}{\partial x^2} + \frac{\partial^2 \tilde{v}}{\partial y^2}) + Re\ (\tilde{u}\frac{\partial v_0}{\partial x} + u_0\frac{\partial \tilde{v}}{\partial x} + \tilde{v}\frac{\partial v_0}{\partial y} + v_0\frac{\partial \tilde{v}}{\partial y} + \frac{\partial \tilde{p}}{\partial y}) \quad (3.36)$$

$$\frac{\partial \tilde{u}}{\partial x} + \frac{\partial \tilde{v}}{\partial y} \quad (3.37)$$

$$\tilde{v}(x_\Gamma, y_\Gamma) \quad (3.38)$$

$$\tilde{u}(0, y) \quad (3.39)$$

$$\tilde{u}(x_{wall}, y_{wall}) \quad (3.40)$$

$$\frac{\partial \tilde{u}}{\partial n}(xmax, y) \quad (3.41)$$

$$\tilde{p}(xmax, ymax) \quad (3.42)$$

Once we have the local linearization FP, we can write the desired estimate for the behavior of F:

$$\begin{aligned} &F(u_0 + \epsilon\tilde{u}, v_0 + \epsilon\tilde{v}, p_0 + \epsilon\tilde{p}) \\ &\qquad = F(u_0, v_0, p_0) \quad + \quad \epsilon\ FP(u_0, v_0, p_0)\ (\tilde{u}, \tilde{v}, \tilde{p}) + O(\epsilon^2). \end{aligned} \quad (3.43)$$

Now we are ready to apply Newton's method to our problem. We evaluate the Navier Stokes function at our starting point, and because our starting point is not a root of F, we get a nonzero right hand side:

$$F(u_0, v_0, p_0) \neq 0. \quad (3.44)$$

How could we get an improved estimate of a root of F? Suppose we drop the ϵ^2 term in the estimate of the behavior of F, assuming that its behavior can be entirely described by the linear part. Then, if there's a nearby point $(u_0 + \epsilon\tilde{u}, v_0 + \epsilon\tilde{v}, p_0 + \epsilon\tilde{p})$, at which F is zero, we must have:

$$\begin{aligned} 0 &= F((u_0 + \epsilon\tilde{u}, v_0 + \epsilon\tilde{v}, p_0 + \epsilon\tilde{p}) && (3.45) \\ &= F(u_0, v_0, p_0) + \epsilon\ FP(u_0, v_0, p_0)\ (\tilde{u}, \tilde{v}, \tilde{p}), && (3.46) \end{aligned}$$

so, in other words, it must be true that

$$\epsilon FP(u_0, v_0, p_0)(\tilde{u}, \tilde{v}, \tilde{p}) = -F(u_0, v_0, p_0). \tag{3.47}$$

If we can solve this equation, then we produce an improved estimate of the root:

$$(u_1, v_1, p_1) = (u_0 + \epsilon\tilde{u}, v_0 + \epsilon\tilde{v}, p_0 + \epsilon\tilde{p}). \tag{3.48}$$

If we find that the function value at this new point is still unacceptably large, we may repeat the refinement process, linearizing around our current estimate and solving the resulting equations for the increments.

The heart of Newton's method is the equation for the increments. We should not be deluded by its simple form. For our case, this equation is actually a system of linear partial differential equations, which are by no means easy to solve. The important point to note here is that we know we can't solve the original problem, $F(u, v, p) = 0$, involving a system of *nonlinear* partial differential equations. Newton's method has offered us a way to get a good estimate of the solution to that problem, if we are willing to solve a series of related, *linear* problems.

The equations for the Newton increments to the Navier Stokes solution are related to the *Oseen equations* [22], which prescribe the behavior of a small perturbation flow (u, v, p) that can be added to a pre-existing flow $(u_{old}, v_{old}, p_{old})$:

$$-(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}) + Re\ (u\frac{\partial u_{old}}{\partial x} + u_{old}\frac{\partial u}{\partial x} + v\frac{\partial u_{old}}{\partial y} + v_{old}\frac{\partial u}{\partial y} + \frac{\partial p}{\partial x}) = 0 \tag{3.49}$$

$$-(\frac{\partial^2 v}{\partial x^2} + \frac{\partial^2 v}{\partial y^2}) + Re\ (u\frac{\partial v_{old}}{\partial x} + u_{old}\frac{\partial v}{\partial x} + v\frac{\partial v_{old}}{\partial y} + v_{old}\frac{\partial v}{\partial y} + \frac{\partial p}{\partial y}) = 0 \tag{3.50}$$

$$\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} = 0 \tag{3.51}$$

We will see the Oseen equations again, when we consider sensitivities.

Chapter 4

DISCRETE FLUID FLOW EQUATIONS

4.1 Introduction

We began the process of analysing our problem by discretizing the flow region and the boundary conditions, that is, making them describable in terms of the values of a few parameters. But while we have discretized our problem, the *answers* to that problem, that is, the solutions of the continuous Navier Stokes equations, discussed in Chapter 3, are still general functions $u(x, y)$, $v(x, y)$ and $p(x, y)$. Except for very special cases, these answers are essentially only describable by giving their values at every point in the flow region, and this is not something that is computable.

We now consider how we discretize our flow equations and their solutions, so that an answer is describable in terms of the values of a finite number of coefficients, and these coefficients can be determined by solving a finite number of equations. Of course, we need to do this in such a way that the answers to the discretized equations can be expected to approximate the answers to the continuous equations.

The closeness of the approximation is dependent on a *discretization parameter h*. We consider constraints on the choice of a class of approximating finite element spaces which will guarantee that as $h \to 0$, the approximate solutions converge to the continuous solution. We then choose the approximating spaces associated with the *Taylor Hood* element.

We then consider the actual form of the finite element equations that replace the original continuous state equations. These will compose a nonlinear algebraic system of considerable dimension. We solve these nonlinear equations using Newton's method, or a hybrid approach involving Picard iteration followed by Newton's method.

Finally, we quote results about the approximation error between the solutions of the continuous and discrete problems. These results assure us that, for small enough h, the results of the discrete problem give good estimates of the behavior of the continuous problem.

4.2 Derivation of the Finite Element Equations

In general, there are no methods of producing the exact solution functions u, v and p for the continuous Navier Stokes problem. The Poiseuille flow problem was a very special case that allowed us to get an exact solution. However, even that simple problem would become insoluble if the form of the inflow were changed, the walls were made irregular in any way, or a small obstacle were placed in the flow.

We want to look at a wide number of Navier Stokes problems, and we need a way of solving them all. Since we can't solve the continuous problem, we now consider the construction of a discrete version of the continuous problem, using the finite element method. We expect that the solution of the discrete problem will approximate the solution of the continuous problem. Given sufficient computational resources, this approximation can be improved as much as desired. The discrete problem can be solved in a straightforward way for the variety of region shapes, boundary conditions, and Reynolds numbers we are interested in.

4.3 Representation of the Solution

We begin the derivation of the finite element equations by making an assumption about the form of the solution functions. For clarity, where a solution to the original, continuous problem is symbolized as (u, v, p), we will symbolize a solution to the approximating problem as (u^h, v^h, p^h). The solution functions for the original continuous problem might have a very elaborate form, but for the approximate problem, we will immediately assume that solutions are constructed using simple linear combinations of a finite set of basis functions to be selected beforehand.

Since u^h and v^h are really coordinate projections of a two-dimensional velocity vector, there are good physical reasons to use the same set of basis functions for both. We represent a typical velocity basis function as $w_i(x, y)$. For technical reasons the pressure cannot be represented with the same set of basis functions, but must use a different set, a typical element of which we will write as $q_i(x, y)$. The pressure basis functions must be chosen in a way that is compatible with the velocity basis functions. We will not explicitly choose the basis functions yet. If we assume that there are N_w velocity basis functions, and N_q pressure basis functions, we may now write out the *finite element representations* of the discretized solutions:

$$u^h(x,y) = \sum_{i=1}^{N_w} u_i^h \, w_i(x,y) \tag{4.1}$$

$$v^h(x,y) = \sum_{i=1}^{N_w} v_i^h \, w_i(x,y) \tag{4.2}$$

$$p^h(x,y) = \sum_{i=1}^{N_q} p_i^h \, q_i(x,y) \tag{4.3}$$

The symbol u^h is used here in two ways: as the name of the solution function, $u^h(x, y)$, and as the name of an entry u_i^h of the coefficient vector. No confusion should arise from this ambiguity, if it is understood that the function and the coefficient vector represent the same information.

The assumption that we can represent solutions of the discretized problem by a linear combination of basis functions makes the discretized problem much simpler than the continuous one. To define the solutions (u^h, v^h, p^h), we simply have to determine the $2N_w + N_q$ coefficients of the basis functions, rather than determining the value at every point (x, y) of three almost arbitrary functions (u, v, p).

4.4 Discretizing the Geometry

We must now produce a set of appropriate basis functions for the problem. To do so, we will first produce a set of *nodes*, that is, special points in the region Ω and on the boundary Γ. We will actually have two sets of nodes, *pressure nodes* and *velocity nodes.* However, for the method we will use, the pressure nodes will simply be a subset of the velocity nodes.

Much thought can go into the choice of the number and location of the nodes. We will make some very simple choices. We will march along the horizontal axis of the channel, taking equally spaced steps. At each step, including the inflow and outflow boundaries, we will place a column of nodes. The nodes in a particular column will be equally spaced in the vertical direction. We require the first node in a column to lie on the bottom wall or bump, and the last node to lie on the upper wall. Each column will have the same number of nodes, but if the column is above the bump, then the vertical spacing between the nodes will be changed from the spacing used in the rectangular portion of the channel. With an eye on future needs, we will require that there be an *odd* number of rows and columns in both directions. For convenience, we will also assume that there is some numbering of the nodes, which we will call the *global node ordering.*

The maximum spacing between nodes will have a certain effect on the accuracy of the

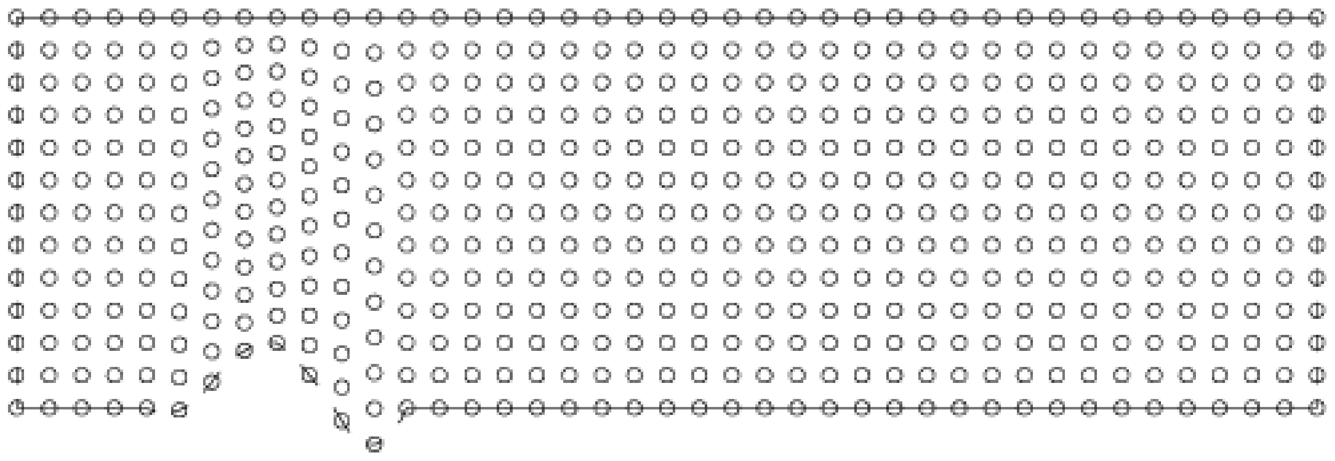

Figure 4.1: Decomposition of a flow region into nodes.

computational results. The value of the discretization parameter h can be taken to be the larger of the two spacings in the rectangular region of the channel, that is, the horizontal and vertical node spacings.

A typical decomposition into nodes is shown in Figure 4.1.

4.5 Choosing a Master Element

In order to generate the basis functions, we are now going to use the nodes to decompose the flow region into a set of separate patches of area, called *elements*. Each element will have the same general structure, and will be defined by listing the nodes that make it up.

There are many choices for elements, but for a particular physical problem, only some elements may be appropriate. Essentially because of the way the the Navier Stokes equations involve two types of state variables, the velocities and pressures, the finite element formulation results in a problem of *mixed type*. This in turn implies that the usual finite element convergence results will not apply unless the basis functions (or equivalently, the approxi-

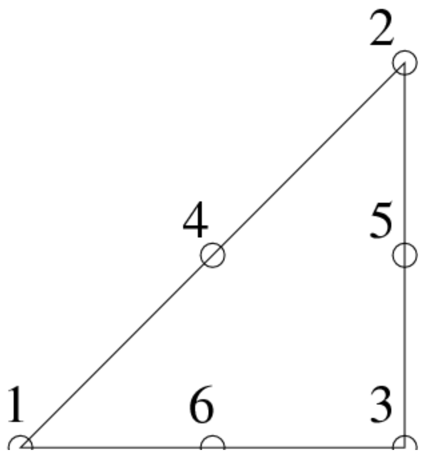

Figure 4.2: A Taylor Hood element.
All nodes have an associated velocity. Nodes 1, 2 and 3
have an associated pressure as well.

mating spaces) for velocities and pressures are chosen in a suitable, compatible way. The main constraint on this choice is known as the *inf-sup* or *Ladyzhenskaya-Babuska-Brezzi* condition. Verification that a particular set of basis functions satisfy this condition is not trivial. We will use the *Taylor Hood* element, whose approximating spaces for velocities and pressure have been shown to satisfy the inf-sup condition. For details, refer to Girault and Raviart [13].

Geometrically, the Taylor Hood element TH has the shape of a triangle, outlined by three *corner nodes* and three *midside nodes*. We will use the special coordinate system (ξ, η) when referring to points in this element. The (ξ, η) coordinates of the corner nodes 1, 2 and 3 are then (0,0), (1,1) and (1,0). All six nodes will be associated with velocity, but only the corner nodes will be associated with pressure.

Details about the construction of the associated basis functions will be discussed in the next section. For the moment, we confine ourselves to certain geometrical issues.

It's easy to see that we can completely cover the rectangular portion of the flow region with copies of the Taylor Hood element, with no overlap. The area over the bump, however, might seem to present us with a problem. On the lower boundary of this area we have the curved surface of the bump. Above the bump, the rectilinear spacing of nodes is disrupted, so that the three nodes needed to form the horizontal side of a Taylor Hood element will not generally lie on a straight line.

We may deal with this more challenging geometry by using the standard technique of *isoparametric* mapping, which allows us to produce smoothly curved images of the Taylor Hood element that fit the local geometry. A simple mapping technique allows us to make the appropriate adjustments for computing basis functions and spatial derivatives on such an isoparametric element.

We are now prepared to produce a *triangulation*, that is, a decomposition of the original flow region into elements with disjoint interiors. We consider each element in the triangulation to be the image of the master element. If we call the i-th element in the triangulation Ω_i, then we can actually construct a differentiable mapping

$$\phi_i : TH \to \Omega_i, \tag{4.4}$$

which maps any point $(\xi, \eta) \in TH$ to a unique point $(x, y) \in \Omega_i$, and so that every point of Ω_i is the image of some point (x, y). Hence, there is also an inverse map:

$$\phi_i^{-1} : \Omega_i \to TH. \tag{4.5}$$

This mapping is completely determined by specifying the correspondence between the 6 nodes in the Taylor Hood element and the 6 nodes in the image. Each node in an element is the image of a node in TH; the *local node index* of a node in Ω_i is the label for the node's preimage in TH. A typical triangulation of a flow region, including isoparametric elements above the bump, is displayed in Figure 4.3.

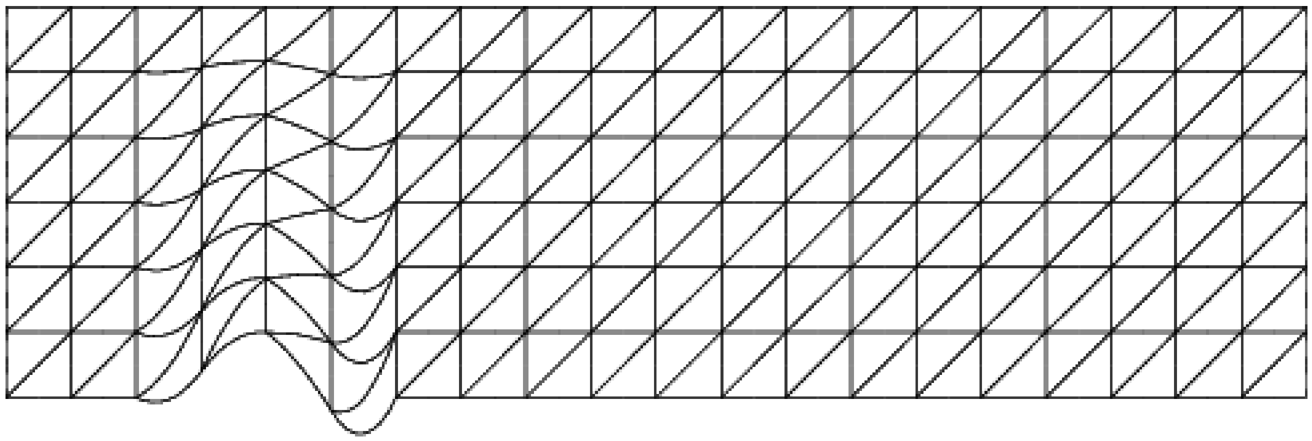

Figure 4.3: Decomposition of a flow region into elements.
These elements are built from the nodes shown in Figure 4.1.

The triangulation of the region in Figure 4.3 is built out of multiple copies of the Taylor Hood element, though some of them may have been curved or stretched a bit by an isoparametric mapping. Because we have decomposed the large, complicated geometry into multiple copies of a single, simple element, and because we know how these simple shapes build the flow region, we can proceed to construct the basis functions for the single master element, and our work will then naturally apply to all the images of that element.

4.6 Constructing Basis Functions

We are now prepared to construct the basis functions that will be used to represent the solution functions. The basis functions will be divided into two families, with $w_i(x, y)$ used for velocities and $q_i(x, y)$ for pressures. We will first consider the velocity basis functions.

Each velocity basis function will be associated with a particular velocity node. These nodes will have several special properties. First, at any velocity node, the associated velocity basis function will have the value 1, while all other velocity basis functions will be zero there.

Secondly, if some function $g^h(x, y)$ is a linear combination of the velocity basis functions, then we can determine the coefficient g_i^h associated with the velocity node i by evaluating g^h at that node. In other words, the finite element coefficients for g^h are its values at the associated nodes.

Let i be the global number of any velocity node, and $w_i(x, y)$ the associated velocity basis function. At any point (x, y) outside of the flow region Ω, we let $w_i(x, y) = 0$. Any point (x, y) within the flow region must lie in some element Ω_j. If node i does not lie on the boundary of element Ω_j, then again $w_i(x, y)$ is zero. We are left with the case where node i lies on the the boundary of Ω_j. In this case, we refer to the local node number $l(i)$ of node i in Ω_j. We then consider the unique quadratic polynomial $\hat{w}_i(\xi, \eta)$ defined on the Taylor Hood element TH, which has the value 1 at node $l(i)$ and 0 at the five other velocity nodes. We then define

$$w_i(x, y) = \hat{w}_i(\phi^{-1}(x, y)). \tag{4.6}$$

This slightly cumbersome definition allows us to consider both nonisoparametric and isoparametric cases together. The "preimage" of the basis function, $\hat{w}_i(\xi, \eta)$, is always a quadratic polynomial in ξ and η. For elements with straight sides, it is also true that $w_i(x, y)$ is quadratic in x and y, basically because ϕ^{-1} is linear for such elements. But in an isoparametric element, the basis function $w_i(x, y)$ will not, in general have the simple form of a quadratic polynomial when written in terms of x and y.

It should be clear from the definition that $w_i(x, y)$ is always 1 at node i and zero at all other velocity nodes, as desired. It should also be clear that the function $w_i(x, y)$ is well defined, even when a point (x, y) lies on the boundary of two or more elements, and that w_i is a continuous function everywhere.

A little thought will reveal the fact that, if we choose an arbitrary vector of values u_i^h, then the function defined by

$$u^h(x,y) = \sum_{i=1}^{N_w} u_i^h \; w_i(x,y) \tag{4.7}$$

will have the value u_i^h at node i.

The pressure basis functions are defined similarly, keeping in mind that in any element there are only three pressure nodes. The major resulting difference for pressure nodes is that we consider the unique *linear* polynomial $\hat{q}_i(\xi, \eta)$ defined on the Taylor Hood element TH, which has the value 1 at node $l(i)$ and 0 at the two other pressure nodes.

In terms of basis functions, a shorthand (and for isoparametric elements, slightly inaccurate) way of defining the Taylor Hood element is, therefore, the use of piecewise quadratic velocities and piecewise linear pressures on a 6 node triangular element.

The use of the finite element basis functions has a number of advantages. In particular, although a discrete set of data represents the solution functions, the solution functions themselves are continuous functions, defined everywhere in the flow region. Moreover, it is obvious how to take any continuously defined function and construct the finite element representation for it, simply by sampling it at the appropriate nodes. (Other versions of the finite element method, such as the least squares approach, do not share this particular feature). Finally, operations such as differentiation of a solution function can be reduced to the same operations on the underlying basis functions, whose properties are easy to derive.

4.7 Discretizing the Navier Stokes Equations

We now have a way of representing functions over the flow region. An arbitrary set of values assigned to the nodes in the proper way will produce a set of functions (u, v, p). However, we don't want an arbitrary set of functions; we want functions that come close to the solution functions for the continuous Navier Stokes equations. How can we do this?

The accepted finite element technique poses the equivalent question of how we determine each of the unknown coefficients. To do so, we go back to the three equations that define the continuous Navier Stokes problem. In particular, let us start with the horizontal momentum equation. For each unknown coefficient u_i^h corresponding to a horizontal velocity, we will multiply the horizontal momentum equation by the corresponding basis function $w_i(x, y)$, and integrate over the region Ω, to arrive at the following equation:

$$\int_\Omega (-\frac{\partial^2 u^h}{\partial x^2} - \frac{\partial^2 u^h}{\partial y^2} + Re(u^h \frac{\partial u^h}{\partial x} + v^h \frac{\partial u^h}{\partial y} + \frac{\partial p^h}{\partial x}))\ w_i\ d\Omega = 0. \tag{4.8}$$

For technical reasons, including the desire to allow the use of basis functions of lower differentiability, we will use integration by parts to lower the order of differentiation on certain terms involving the solution variables. To do so, we apply a version of *Green's theorem*:

Theorem 4.1 (Green's Theorem in the Plane) *Suppose Ω is a bounded, open, connected subset of R^2 with a piecewise differentiable boundary Γ. Suppose that u and w are functions defined on $\bar{\Omega}$, the closure of Ω, and continuously differentiable there. Then*

$$\int_\Omega \frac{\partial u(x,y)}{\partial x} w(x,y)\ dx\ dy = -\int_\Omega u(x,y) \frac{\partial w(x,y)}{\partial x}\ dx\ dy + \int_\Gamma u(x,y) w(x,y) \hat{n}_x(x,y)\ ds \tag{4.9}$$

where $\hat{n}_x(x, y)$ is the x component of the unit outward normal vector to Γ. A similar formula holds for differentiation with respect to y.

For the horizontal momentum equation, we wish to apply Theorem 4.1 to the terms $-\frac{\partial^2 u^h}{\partial x^2} w_i$ and $-\frac{\partial^2 u^h}{\partial y^2} w_i$, reducing the highest order of differentiation applied to a velocity basis function from 2 to 1. We could also apply Green's theorem to the term $Re \frac{\partial p^h}{\partial x} w_i$, eliminating all differentiation of pressures, at the cost of some unwieldy boundary terms.

If we carry out this same set of operations on the other two governing equations, we arrive at the *discretized finite element equations*:

$$\int_\Omega (\frac{\partial u^h}{\partial x}\frac{\partial w_i}{\partial x} + \frac{\partial u^h}{\partial y}\frac{\partial w_i}{\partial y} + Re(u^h\frac{\partial u^h}{\partial x} + v^h\frac{\partial u^h}{\partial y} + \frac{\partial p^h}{\partial x})w_i)\, d\Omega = \int_\Gamma \frac{\partial u^h}{\partial n} w_i\, ds \tag{4.10}$$

$$\int_\Omega (\frac{\partial v^h}{\partial x}\frac{\partial w_i}{\partial x} + \frac{\partial v^h}{\partial y}\frac{\partial w_i}{\partial y} + Re(u^h\frac{\partial v^h}{\partial x} + v^h\frac{\partial v^h}{\partial y} + \frac{\partial p^h}{\partial y})w_i)\, d\Omega = \int_\Gamma \frac{\partial v^h}{\partial n} w_i\, ds \tag{4.11}$$

$$\int_\Omega (\frac{\partial u^h}{\partial x} + \frac{\partial v^h}{\partial y})q_i\, d\Omega = 0 \tag{4.12}$$

In the boundary integrals, we should point out the meaning of the expression $\frac{\partial u^h}{\partial n}$, which is sometimes called the *normal derivative*:

$$\frac{\partial u^h}{\partial n} \equiv \frac{\partial u^h}{\partial x}\hat{n}_x + \frac{\partial u^h}{\partial y}\hat{n}_y, \tag{4.13}$$

where, as before, the outward unit normal vector to the flow region Ω at (x, y) is the vector $(\hat{n}_x(x, y), \hat{n}_y(x, y))$.

Now the unknowns in this system seem to be the functions $u^h(x, y)$, $v^h(x, y)$, and $p^h(x, y)$, but since we assume that these functions are represented by linear combinations of the basis functions, the true unknowns are the coefficients, that is, the numbers u_i^h, v_i^h, and p_i^h. We could make this explicit by replacing all occurrences of the functions $u^h(x, y)$, $v^h(x, y)$, and $p^h(x, y)$ in the finite element equations by the finite sums, but the resulting expressions are much too cumbersome to work with.

By construction, there are exactly as many of these equations as there are unknown coefficients u_i^h, v_i^h and p_i^h. These equations are *nonlinear*, as are the original Navier Stokes equations, but they are strictly algebraic (in fact, they are actually *quadratic*). That means that they may be treated by a form of Newton's method in which the linearized operator to be inverted is simply a Jacobian matrix.

4.8 Discretizing the Boundary Conditions

In order to have a completely posed discretized problem, we must account for the boundary conditions that were applied in the continuous case. We will attempt to do this in such a way that we eliminate redundant equations. In this process, we will disrupt somewhat the original situation, in which, for instance, every corner node always had two associated unknown velocity components and an unknown pressure.

The boundary conditions for the continuous case were:

$$\begin{aligned} v(x,y) &= 0 \text{ along the boundary;} \\ u(0,y) &= Inflow(y,\lambda) \text{ along the inflow;} \\ u(x,y) &= 0 \text{ on the walls and the bump;} \\ p(xmax,ymax) &= 0. \end{aligned}$$

Note, however, that we do *not* have to impose the boundary condition

$$\frac{\partial u}{\partial x}(xmax,y) = 0 \text{ along the outflow,} \tag{4.14}$$

because this is automatically enforced weakly by the finite element method itself.

As a first step towards discretization, we may simply interpret these conditions as being statements about the value of the state solution at the corresponding velocity or pressure nodes. In other words, the discrete boundary conditions may be interpreted as:

$$\begin{aligned} v^h(x,y) &= 0 \text{ at all velocity nodes on } \Gamma; && (4.15) \\ u^h(0,y) &= Inflow^h(y,\lambda) \text{ at all inflow velocity nodes;} && (4.16) \\ u^h(x,y) &= 0 \text{ at all velocity nodes on the walls and the bump;} && (4.17) \\ \frac{\partial u^h}{\partial x}(xmax,y) &= 0 \text{ at outflow velocity nodes;} && (4.18) \\ p^h(xmax,ymax) &= 0 \text{ for the upper right pressure node.} && (4.19) \end{aligned}$$

In the above equations, $Inflow^h(y, \lambda)$ denotes that discretization has been applied to our original inflow function. In our case, we will use the interpolant, that is, at every velocity node, it will be true that:

$$Inflow^h(y, \lambda) = Inflow(y, \lambda) \tag{4.20}$$

The discretized boundary conditions may be translated into algebraic equations involving the finite element coefficients, and we may append these equations to the discretized state equations. However, most of these boundary conditions are simple enough that from them we can easily determine the values of certain coefficients. It seems wasteful to treat such quantities as unknowns. We will find that, by proper application of the given boundary conditions, we can solve for many of the unknowns along the boundary. For such eliminated unknowns, we can discard the corresponding finite element equations and basis functions from our system.

Let us consider the vertical velocity boundary condition, $v^h(x, y) = 0$ at all velocity nodes on Γ. At each velocity node i along the boundary, we know that the value of the associated finite element coefficient v_i^h is zero; in other words, v_i^h is *not* an unknown. We may therefore apply the boundary condition by dropping the discrete finite element equation associated with v_i^h, which would normally be used to solve for the unknown coefficient. In all other equations we replace occurrences of v_i^h by 0. Node i will no longer have an associated vertical velocity.

We handle the Dirichlet boundary conditions on the horizontal velocity at the inflow, along the walls and on the bump, and the condition on the pressure in the upper right corner in a similar way.

Since we have already eliminated the velocity unknowns associated with the walls, bump, and inflow, all remaining velocity basis functions w_i are zero on Γ, except, possibly, along the outflow boundary. But, as we have pointed out, the finite element method automatically weakly enforces the boundary conditions:

$$\frac{\partial u^h}{\partial x} = \frac{\partial u^h}{\partial n} = 0 \frac{\partial v^h}{\partial x} = \frac{\partial v^h}{\partial n} = 0 \tag{4.21}$$

which means that in Equations (4.10) and (4.11) the right hand side integrals over Γ drop out, leaving us with the slightly simpler system:

$$\begin{aligned}
\int_\Omega (\frac{\partial u^h}{\partial x}\frac{\partial w_i}{\partial x} + \frac{\partial u^h}{\partial y}\frac{\partial w_i}{\partial y} &\quad + \quad Re(u^h\frac{\partial u^h}{\partial x} + v^h\frac{\partial u^h}{\partial y} + \frac{\partial p^h}{\partial x})w_i)\; d\Omega = 0 &(4.22)\\
\int_\Omega (\frac{\partial v^h}{\partial x}\frac{\partial w_i}{\partial x} + \frac{\partial v^h}{\partial y}\frac{\partial w_i}{\partial y} &\quad + \quad Re(u^h\frac{\partial v^h}{\partial x} + v^h\frac{\partial v^h}{\partial y} + \frac{\partial p^h}{\partial y})w_i)\; d\Omega = 0 &(4.23)\\
\int_\Omega (\frac{\partial u^h}{\partial x} + \frac{\partial v^h}{\partial y})q_i\; d\Omega &\quad = \quad 0 &(4.24)
\end{aligned}$$

Because the inflow velocities are generally nonzero, the treatment of the boundary conditions means that the previously unknown velocity coefficient u_i^h at an inflow node is replaced by a known constant. In some cases, this will result in the computation of a term which we will move to the right hand side.

4.9 Estimating the Solution at Nearby Parameters

Let us suppose, for a moment, that the definition of the discretized Navier Stokes problem we are solving includes a set of parameters β, as in Chapter 2, and that we can consider the corresponding flow solutions to be functions of those parameters. If we wish to keep this relationship in mind, we write the flow solution as $(u^h(\beta), v^h(\beta), p^h(\beta))$.

Now our discrete Navier Stokes equations and boundary conditions comprise a system of nonlinear algebraic equations, which we plan to solve using some iterative scheme. The very first time that we try to solve the nonlinear system, say at the parameter values β_0, we'll use zero as the starting guess for $(u^h(\beta_0), v^h(\beta_0), p^h(\beta_0))$. But if we are able to solve that initial system, then if we require the values of (u^h, v^h, p^h) at a nearby set of parameters, we should surely be able to produce a better starting point for the next iteration than simply the zero guess. One way to get a better estimate might be to use the previous solution:

$$u^h(\beta_0 + \Delta\beta) \approx u^h(\beta_0), \tag{4.25}$$

but we usually have data that approximates the first partial derivatives such as $\frac{\partial u^h(\beta)}{\partial \beta_i}$, since we are computing the discretized sensitivities or finite difference estimates of the sensitivities. Hence, we can make an even better estimate by using the first order *Euler prediction*:

$$u^h(\beta_0 + \Delta\beta) \approx u^h(\beta_0) + \sum_i \frac{\partial u^h(\beta_0)}{\partial \beta_i} \Delta\beta_i. \tag{4.26}$$

4.10 Picard Iteration Applied to the Discrete Problem

In some cases, the starting point produced by the Euler prediction will be close enough to the correct solution that Newton's method can converge. But this may not be so, particularly for higher Reynolds numbers. Thus, in some cases, we must work harder to produce a satisfactory starting point.

In such situations, we will use the Euler estimate as the starting point for a Picard iteration scheme. We will use the following decomposition $H(X, X_{old})$ of the discrete state equations $F(X)$:

$$\begin{aligned}
\int_\Omega (\frac{\partial u^h}{\partial x}\frac{\partial w_i}{\partial x} + \frac{\partial u^h}{\partial y}\frac{\partial w_i}{\partial y} &+ Re(u^h_{old}\frac{\partial u^h}{\partial x} + v^h_{old}\frac{\partial u^h}{\partial y} + \frac{\partial p^h}{\partial x})w_i)\, d\Omega = 0 && (4.27)\\
\int_\Omega (\frac{\partial v^h}{\partial x}\frac{\partial w_i}{\partial x} + \frac{\partial v^h}{\partial y}\frac{\partial w_i}{\partial y} &+ Re(u^h_{old}\frac{\partial v^h}{\partial x} + v^h_{old}\frac{\partial v^h}{\partial y} + \frac{\partial p^h}{\partial y})w_i)\, d\Omega = 0 && (4.28)\\
\int_\Omega (\frac{\partial u^h}{\partial x} + \frac{\partial v^h}{\partial y})q_i\, d\Omega &= 0 && (4.29)
\end{aligned}$$

along with the boundary conditions (4.15) through (4.19). Given the value of X_{old} =

$(u^h_{old}, v^h_{old}, p^h_{old})$ the equations $H(X, X_{old}) = 0$ are a linear system solvable for the new coefficients $X = (u^h, v^h, p^h)$ that define the next approximation to the flow solution.

The following theorem from Rheinboldt and Ortega [24] suggests how our Picard iteration could be analyzed and modified to guarantee global convergence:

Theorem 4.2 (Global Convergence of Picard Iteration) *Let $A \in L(R^n)$ be a symmetric and positive definite matrix, assume that $\phi : R^n \to R^n$ is continuously differentiable, that $\phi'(x)$ is symmetric and positive semidefinite for all $x \in R^n$, and that there is a constant $\beta < +\infty$ for which*

$$\|\phi'(x)\|_2 \leq \beta, \forall x \in R^n. \tag{4.30}$$

Then the equation $Ax + \phi(x) = 0$ has a unique solution x^*, and for any $x_0 \in R^n$, the sequence $\{x_k\}$ defined by the Picard iteration:

$$(A + (\beta/2)I)x_{k+1} = (\beta/2)x_k - \phi(x_k) \tag{4.31}$$

converges to x^*.

The version of Picard iteration we are currently employing does not attempt to estimate the value of the constant β, and thus has the form

$$Ax_{k+1} = -\phi(x_k) \tag{4.32}$$

which is not guaranteed to converge. Indeed, we have seen cases where our Picard formulation fails in practice. In the future, we intend to investigate the reformulation of our iteration along the lines suggested by the theorem, so that we may guarantee global convergence.

Picard iteration can be particularly useful at higher Reynolds numbers, where the radius of convergence for Newton's method is generally much decreased. However, at least when the hypotheses of Theorem 4.2 apply, the guaranteed convergence is only of linear order. Thus, a reasonable use of Picard iteration is simply to reduce the residual of the initial point enough that the Newton method, with its superior quadratic convergence, may be applied to bring the iteration to a rapid conclusion.

With our current Picard iteration, which does not satisfy the hypotheses of Theorem 4.2, we cannot guarantee global convergence. Thus, our iteration may fail. It is sometimes possible to recover from such a failure; for instance, if we are computing a solution at a high Reynolds number, we can retry the computation by finding a solution at a low Reynolds number, and then using the Euler prediction method discussed in the previous section to produce a better starting estimate at the Reynolds number of interest. In other cases, we may be able to recover by reducing the mesh parameter h, although this is a costly solution.

4.11 The Newton Method Applied to the Discrete Problem

Assuming we have a starting point X_0 that is sufficiently close to a solution of $F(X) = 0$, Newton's method produces a sequence of iterates by computing a correction ΔX to the current iterate. The heart of the iteration is the pair of calculations:

$$\Delta X_i \leftarrow -FP(X_i)^{-1} F(X_i) \tag{4.33}$$

$$X_{i+1} \leftarrow X_i + \Delta X_i \tag{4.34}$$

The expensive part of the calculation is the computation of the Jacobian matrix FP and its factorization. Many attempts to improve the efficiency of the Newton method begin by trying to reduce the storage used for this matrix or the frequency with which it is calculated.

As soon as we have the new iterate X_{i+1}, we can compute the residual $F(X_{i+1})$. The norm of this quantity is an indication of how far away we are from a solution. Thus, if we monitor this quantity, its behavior will allow us to decide whether to take another iterative step, or to accept X_{i+1} as our approximation to the solution, or to determine that the iteration has failed to converge, or is actually diverging.

The local behavior of Newton's method is laid out in the following theorem (Dennis and Schnabel, [10]):

Theorem 4.3 (Local Convergence of Newton's Method in R^n) *Suppose that the function $F : R^n \rightarrow R^n$ is continuously differentiable in an open convex set $\mathcal{D}$, and that there is a point $x^* \in \mathcal{D}$ at which $F(x^*) = 0$. Suppose that the Jacobian matrix $FP(x^*)$ is invertible, with $\|FP^{-1}(x^*)\| = \beta$. Suppose, finally, that there is an open ball $B_r(x^*)$ of radius r centered at x^*, with $B_r(x^*) \subset \mathcal{D}$, so that for every $x \in B_r(x^*)$, $FP(x)$ is Lipschitz continuous, with Lipschitz constant γ independent of x.*

Then there is an $\epsilon > 0$ with the property that, for any starting point $x_0 \in B_\epsilon(x^*)$, the sequence of Newton iterates $\{x_k\}$ is well defined, converges to x^*, and for each k satisfies

$$\|x_{k+1} - x^*\| \leq \beta\gamma\|x_k - x^*\|^2. \tag{4.35}$$

This theorem informs us that if we are close enough to the correct solution, Newton's method will converge quadratically. Conversely, practical experience shows that if the starting point is too far away, Newton's method will diverge. There is no general way to compute this critical distance beforehand; therefore, we must expect from time to time that we must deal with a Newton failure. In some cases, it may be possible to recover by producing a better starting guess, or trying to solve an easier problem (one whose parameters are closer to the parameters of a problem we've already solved). However, some failures of Newton's method will cause the entire optimization to fail.

To implement Newton's method, we must differentiate each finite element equation with respect to the unknown coefficients, a tedious computation. For the state equations (4.22)-(4.24) associated with basis functions w_i or q_i, we list the entries in the Jacobian FP with respect to a typical coefficient u_j, v_j or p_j. These entries are evaluated at the "old" solution point, $(u^h_{old}, v^h_{old}, p^h_{old})$, and are used to solve for the change in the coefficients that will give us the next Newton approximant.

The derivative of the horizontal momentum equation associated with basis function w_i with respect to coefficient u^h_j is:

$$\int_\Omega (\frac{\partial w_j}{\partial x}\frac{\partial w_i}{\partial x} + \frac{\partial w_j}{\partial y}\frac{\partial w_i}{\partial y} + Re(w_j \frac{\partial u^h_{old}}{\partial x} + u^h_{old}\frac{\partial w_j}{\partial x} + v^h_{old}\frac{\partial w_j}{\partial y})w_i)\; d\Omega; \tag{4.36}$$

the derivative of the horizontal momentum equation associated with basis function w_i with respect to coefficient v^h_j is:

$$\int_\Omega Re\; w_j \frac{\partial u^h_{old}}{\partial y} w_i\; d\Omega; \tag{4.37}$$

the derivative of the horizontal momentum equation associated with basis function w_i with respect to coefficient p^h_j is:

$$\int_\Omega Re\; \frac{\partial q_j}{\partial x} w_i\; d\Omega; \tag{4.38}$$

the derivative of the vertical momentum equation associated with basis function w_i with respect to coefficient u^h_j is:

$$\int_\Omega Re\; w_j \frac{\partial v^h_{old}}{\partial x} w_i\; d\Omega; \tag{4.39}$$

the derivative of the vertical momentum equation associated with basis function w_i with respect to coefficient v^h_j is:

$$\int_\Omega (\frac{\partial w_j}{\partial x}\frac{\partial w_i}{\partial x} + \frac{\partial w_j}{\partial y}\frac{\partial w_i}{\partial y} + Re(u^h_{old}\frac{\partial w_j}{\partial x} + w_j \frac{\partial v^h_{old}}{\partial y} + v^h_{old}\frac{\partial w_j}{\partial y})w_i)\; d\Omega; \tag{4.40}$$

the derivative of the vertical momentum equation associated with basis function w_i with respect to coefficient p^h_j is:

$$\int_\Omega Re\; \frac{\partial q_j}{\partial y} w_i\; d\Omega; \tag{4.41}$$

the derivative of the continuity equation associated with basis function q_i with respect to coefficient u_j^h is:

$$\int_\Omega \frac{\partial w_j}{\partial x} q_i \, d\Omega; \tag{4.42}$$

and the derivative of the horizontal momentum equation associated with basis function q_i with respect to coefficient v_j^h is:

$$\int_\Omega \frac{\partial w_j}{\partial y} q_i \, d\Omega. \tag{4.43}$$

We note that the Jacobian that defines the Newton iteration is *not* the same as the system matrix used for Picard iteration, although they have the same structure, and share most of the same entries. This means that it will not be possible to reuse a Picard iteration matrix during the Newton iteration, for instance, a fact that will reduce the options for efficient execution later on.

4.12 The Approximation of the Continuous Problem

We have shown how the discrete problem is derived from the continuous problem. While it is not clear how to solve the continuous problem, we have outlined an algorithm for producing the solution of the discrete problem. We would like, therefore, to determine in what sense the solution of the discrete problem is a good approximation to the solution of the continuous problem.

It is shown by, for instance, Boland and Nicolaides [1], that as long as the solution of the continuous problem $(\mathbf{u}, p)$ has a particular smoothness, then the error in the approximation can be bounded in terms of the mesh parameter h.

Before we can present their results, we must develop the appropriate notation. First, we define the following spaces of functions:

$$L^2(\Omega) \equiv \{f : \Omega \to R : \int_\Omega f^2 \, d\Omega < \infty\}. \tag{4.44}$$

We then define the following related spaces:

$$L_0^2(\Omega) \equiv \{f \in L^2(\Omega) : \int_\Omega f d\Omega = 0\}, \tag{4.45}$$

and the *Sobolev space*:

$$H^k(\Omega) \equiv \{f \in L^2(\Omega) : D^s f \in L^2(\Omega), \text{ s=1,...,k}\} \tag{4.46}$$

where $D^s f$ denotes any derivative of order s.

We define a space of functions that is zero on the boundary of Ω:

$$H_0^k(\Omega) \equiv \{f \in H^k(\Omega) : f = 0 \text{ on } \Gamma\}. \tag{4.47}$$

We also define the following norm on the space L^2 (and, of course, on all its subspaces):

$$\|f\|_0 \equiv (\int_\Omega f^2 d\Omega)^{\frac{1}{2}}, \tag{4.48}$$

as well as the following seminorm on functions in $H_0^1(\Omega)$:

$$|f|_1 \equiv (\sum_{i=1}^n \|\frac{\partial f}{\partial x_i}\|_0^2)^{\frac{1}{2}}. \tag{4.49}$$

We have written $\mathbf{f}$ when a function takes values in R^n, and in such cases, we define the corresponding spaces $\mathbf{H}_0^k(\Omega)$ and so on.

We can now paraphrase the approximation theorem, which states that, for $m = 2$ or 3, if the solutions $\mathbf{u}$ and p to the continuous Navier Stokes problem satisfy:

$$\mathbf{u} \in \mathbf{H}^m(\Omega) \cap \mathbf{H}_0^1(\Omega), \tag{4.50}$$

$$p \in H^{m-1}(\Omega) \cap L_0^2(\Omega), \tag{4.51}$$

then the following error estimates, which compare the solutions of the continuous problem to the solutions $\mathbf{u}^h$ and p^h of the discretized problem, hold uniformly in h:

$$|\mathbf{u} - \mathbf{u}^h|_1 = O(h^{m-1}) \tag{4.52}$$

$$\|\mathbf{u} - \mathbf{u}^h\|_0 = O(h^m) \tag{4.53}$$

$$\|p - p^h\|_0 = O(h^{m-1}) \tag{4.54}$$

For our problem, the exact velocities $\mathbf{u}$ can not, in general, be supposed to lie in $\mathbf{H}^3(\Omega)$, because our region is not convex. The actually smoothness of the solution depends strongly on the amount of deviation from convexity, most prominently in the angles made where the bump surface joins the bottom of the channel. For the class of problems we have chosen, we can expect that the exact velocities will be elements of $\mathbf{H}^2(\Omega)$. This means that, if we can satisfy the other hypotheses of the theorem, that we can expect $O(h^2)$ accurate approximation of velocity, and $O(h)$ accurate approximation of pressure.

On the face of it, the solutions $\mathbf{u}$ and p of the continuous problem do not satisfy the other conditions necessary to apply the results. This is because we do not have zero boundary conditions on $\mathbf{u}$, nor do we require that p have a zero integral over the region. However, these difficulties are easily overcome.

The discrepancy in the pressure conditions is truly minor. If we replace the original pressure condition by

$$\int_\Omega p(x,y)\ d\Omega = 0, \tag{4.55}$$

we will recompute the original result, plus or minus a constant. The choice between the two boundary conditions is strictly a matter of convenience.

For the velocity boundary conditions, we appeal to the discussion in Gunzburger [15], where it is shown that the error estimate given for the homogeneous boundary condition still holds if a nonhomogeneous condition is given, assuming the discretized problem uses the interpolant of the original boundary condition function.

These results tell us that the solution to the finite element problem is indeed close to the solution to the original continuous problem, and that the error of approximation goes to zero with h.

Spatial derivatives of $\mathbf{u}$ and p, which are also produced by the finite element method, are less accurately computed. In general, each degree of spatial differentiation reduces the order of accuracy of approximation by one power of h. We will find later that, in order to produce certain quantities called geometric sensitivities, we will want accurate approximation of the derivatives of the continuous solution such as $u_y \equiv \frac{\partial u}{\partial y}$. We can see that, given the smoothness assumptions about the solution, we can expect a lower order approximation of u_x and u_y:

$$(\|u_x - u_x^h\|_0^2 + \|u_y - u_y^h\|_0^2)^{\frac{1}{2}} = O(h), \tag{4.56}$$

which we will take to mean that we have $O(h)$ error in either approximate derivative u_x^h or u_y^h. This fact will have ramifications on the accuracy of the discretized sensitivity computations.

Chapter 5

STATE SENSITIVITIES WITH RESPECT TO A PARAMETER

5.1 Introduction

As long as we confine ourselves to the feasible set, each set of values for the parameters defined in Chapter 2 defines a corresponding fluid flow problem. From Chapter 3, we may expect that there is a corresponding solution of the problem, and from Chapter 4 we know that, for the same set of parameters, there is also a discretized flow problem whose solution may be computed, and which approximates the solution of the continuous problem. These facts are summarized by stating that, for each set of feasible parameter values, β, there is (at least) one corresponding continuous flow solution (u, v, p), and discretized flow solution (u^h, v^h, p^h).

We would like, if possible, to regard these flow solutions as actual *functions* of the parameters. Despite the fact that we know that solutions to the continuous flow problem are not unique, we will find that, away from certain critical Reynolds numbers, the flow solutions form continuous branches of solutions, which are "locally unique".

Once we regard the flow solutions as *functions* of the parameters, it is natural to investigate the influence of each problem parameter on the components of the solution, a concept called the *sensitivity* of the solution with respect to the parameter.

We show how the state equations that define the state variables may be differentiated to yield equations for the sensitivity of the state variables with respect to a given problem parameter. This operation may be carried out on continuous or discrete state equations.

Since discrete state equations are generally derived from a continuous state equation, we might consider applying the same discretization process to the sensitivities of the continuous equation, and consider how the resulting *discretized sensitivities* are related to the sensitivities of the discrete state equation.

We work through a simple linear differential equation with a parameter, computing the sensitivities and discretized sensitivities, and showing that in this case discretization and differentiation may be interchanged; that is, they *commute.* These operations are not actually guaranteed to commute in all cases, and we briefly mention the case of parameters that affect the problem geometry. The consequences of non-commutativity are discussed in later chapters.

5.2 Questions About the Solutions of Parameterized Problems

In many fluid problems, there are a number of physical quantities which influence the problem, and hence the solution. These quantities might include the viscosity, the strength of the inflow, the temperature, the roughness of the walls, and so on. Especially when the item can be described by a few numerical values, we call such a varying quantity a flow parameter. Some flow parameters might be under the explicit control of the experimenter, while some might represent physical conditions which can't be controlled, but may be measured beforehand.

A special class of flow parameters is called *geometric* or *shape parameters.* Such parameters affect the shape of the flow region, or the location or shape of obstacles in the flow, or the location at which boundary conditions are imposed. The bump parameters that we discussed in Chapter 2 are an example of geometric parameters. As we proceed to the optimization problem we are ultimately interested in, we will see that geometric parameters introduce special complications into the problems we study. We will handle these difficulties at the appropriate time.

We will suppose, however, that if the values of the relevant parameters are known, then the particular fluid flow problem is completely defined, and can be solved to produce at least one flow solution. Let us consider some generic set of one or more parameters which we will call β. We will let u momentarily represent all three flow solution quantities for the continuous problem. If we fix all other aspects of the fluid flow problem, then we would like to regard the flow solution as a function of the parameters that are varying, a relationship we would represent by writing $u(\beta)$. Although we will frequently use such expressions, they are not quite proper. We have already discussed the fact that for higher Reynolds numbers, there may be multiple solutions of a given flow problem, so that the notation $u(\beta)$, suggesting a unique solution, is not strictly correct.

However, there is a sort of “local uniqueness” result for solutions to the flow problem. That is, there are certain generally isolated values of the Reynolds number, where the nature of the solution changes, and the graph of the solution may bifurcate. If we remove the data at those critical Reynolds numbers, the remaining solution graph breaks apart into a collection of open sets, each of which is a smoothly varying function of the Reynolds number, called a *smooth solution branch.* In general, this means that for a given set of parameters β_0, if we compute a flow solution u_0, we may reasonably write $u(\beta)$ to represent the local family of flow solutions that lie along the open branch that passes through the given solution $u(\beta_0) \equiv u_0$.

Once we confine our attention to a smooth branch of solutions, we may pose certain natural questions. We phrase these questions in terms of the continuous problem, though we will be interested in the discretized problem as well. Questions we might ask about the relationship between u and β include:

- Suppose that, for some set of parameter values β_0, we have computed a particular solution u_0. Can we estimate the value of flow solutions for nearby values of β?
- Can we, in fact, guarantee that there *is* a solution at a nearby value of β?
- Can we guarantee that the solution is *unique*?
- If we have the value $J(u_0)$ of some functional J at u_0, can we estimate the value of J at flow solutions for nearby values of β?
- At the given solution u_0, can we estimate the value of $\dfrac{\partial J}{\partial \beta}$?

We will show that the answer to these questions is generally positive, and that the primary tools we need are the linearizations (or, for the discrete case, the partial derivatives) of the state equations with respect to u and β, and the partial derivatives of the state variable u with respect to the parameters β, called the solution *sensitivity*.

5.3 The Sensitivity of a Continuous Solution

Let us assume that we have a solution space of pairs $((u, v, p), \beta)$ of state variable functions (u, v, p) and parameter sets $\beta \in R^m$. We assume that each state function is a suitably differentiable function defined over the open domain $\Omega \subset R^p$ and taking values in R^q. We assume that Ω has a piecewise smooth boundary Γ, and we allow the possibility that the domain Ω depends on β.

We assume that there is a state function, written $F((u,v,p),\beta) = 0$, with $u, v \in C^2(\Omega)$, $p \in C^1(\Omega)$, and $\beta \in R^n$. The function $F((u,v,p),\beta)$ comprises internal constraints, which are applied at all points $x \in \Omega$, and boundary constraints that hold just at points $x \in \Gamma$. A solution of the parameterized Navier Stokes equations may then be considered as a pair $((u,v,p),\beta)$ for which the equation $F((u,v,p),\beta) = 0$ holds.

Suppose we have already found a solution to the state equation, that is, a pair $((u_0, v_0, p_0)\beta_0)$ from the solution space, which satisfy the state equations. Having found such a solution, we are interested in understanding how the two components (u,v,p) and β are related, and whether, for small changes in β_0, we can make correspondingly small changes in (u_0, v_0, p_0) and come up with a new pair that again solves the state equation.

We would like to assert the existence of an implicitly defined function $h(\beta)$ which allows us to view the state solution locally as an explicit function of β. The function $h(\beta)$, if it exists, should have the property that it solves the flow problem for the parameters β, that is, for β "near" to β_0:

$$F(h(\beta),\beta) = 0 \ . \tag{5.1}$$

Moreover, we would like $h(\beta)$ to be continuous in β, and in fact continuously differentiable. In that case, assuming that the state system F has a continuous Frechet derivative, we may differentiate Equation (5.1) with respect to any component parameter β_i to arrive at:

$$F_{uvp}(h(\beta),\beta)\ h_{\beta_i}(\beta) + F_{\beta_i}(h(\beta),\beta) = 0 \ . \tag{5.2}$$

If the function $h(\beta)$ exists, then we may abuse notation and write $(u(\beta), v(\beta), p(\beta))$ instead, to emphasize the idea that, locally at least, the flow solution may be regarded as a function of the parameters. In that case, instead of $h_{\beta_i}(\beta)$ we write $(u_{\beta_i}(\beta), v_{\beta_i}(\beta), p_{\beta_i}(\beta))$.

We call Equation (5.2) the *continuous sensitivity equation*. We call the quantity $u_{\beta_i}(\beta)$ the *continuous sensitivity* of the solution component $u(\beta)$ with respect to the parameter β_i, although this is, of course, simply another name for the partial derivative of $u(\beta)$ with respect to β_i.

5.4 The Sensitivity of a Discrete Solution

When we talk about the solution of a discrete state equation, we are usually referring to the solution of a *discretized* continuous state equation. That is, it is understood that the discrete state equation was derived, via discretization, from some continuous state equation. This will be the case for all the problems we consider.

Whether or not there is a corresponding continuous state equation, we will assume that the discrete state equations do *not* involve any differentiation or integration with respect to the parameters, but rather are purely algebraic in u^h, v^h, p^h and β.

In fact, the finite element equations we have been considering are fairly simple functions of the unknown coefficients u_i^h, v_i^h and p_i^h. It might seem that the discretized finite element equations, Equations (4.10)-(4.12) plus the boundary conditions, do not have an algebraic form, since they involve integrals and derivatives of the state variables with respect to the spatial variables. But since the basis functions and approximating functions used in the finite element method are piecewise polynomials, the integrals such as

$$\int_\Omega Re\ u^h \frac{\partial u^h}{\partial x} w_i \, d\Omega \ , \tag{5.3}$$

are integrals of polynomials. This term, in particular, is equivalent to the expression:

$$\sum_{j=1}^{N_w} \sum_{k=1}^{N_w} \left(\int_\Omega Re\ w_j \frac{\partial w_k}{\partial x} w_i \, d\Omega \right) u_j^h u_k^h \ , \tag{5.4}$$

where the quantity in parentheses is simply a numerical quantity to be determined.

Because of complications that may occur with isoparametric elements, or nonpolynomial source terms, or boundary conditions, these integrals are usually carried out using numerical quadrature. Nonetheless, it should be clear now that the discretized finite element equations are merely a set of *quadratic* equations in the finite element coefficients.

We begin consideration of the discrete case by quoting a version of the *implicit function theorem for R^n*. This theorem gives us conditions under which a differentiable *relationship* involving state variables and parameters guarantees that there is actually an implicitly defined local differentiable *function* for the state variables in terms of the parameters.

Theorem 5.1 (The Implicit Function Theorem for R^n) *Suppose $f : \mathcal{D} \rightarrow R^n$, where $\mathcal{D} \subset R^n \times R^m$ is an open set, and that $f \in C^1(\mathcal{D})$, that $f(x_0, y_0) = 0$ for some $(x_0, y_0) \in \mathcal{D}$, and that rank$[f_x](x_0, y_0) = n$.*

Then there exists an open set $U \subset R^n \times R^m$, containing (x_0, y_0), an open set $V \subset R^m$ containing y_0, and a function $h : V \rightarrow R^n$ such that $h(y_0) = x_0$ and $f(h(y), y) = 0$ for all $y \in V$.

Furthermore, h is uniquely determined by $(h(y), y) \in U$ for all $y \in V$, and $h \in C^1(V)$ and $h'(y)$ satisfies:

$$f_x(h(y), y)h'(y) + f_y(h(y), y) = 0. \tag{5.5}$$

For our purposes, we want to slightly modify the hypotheses of this theorem, and change the terminology to fit our own more closely. In particular, we replace the rank condition by a stronger condition on the solvability of a certain linear system. This is in part because in order to get the "base solution" $((u_0^h, v_0^h, p_0^h), \beta_0)$, we will have to solve exactly such a linear system, and in part because such a formulation is closer to that used for the continuous problem. In the following theorem, we will commit a slight abuse of notation, by writing

$u^h(\beta)$ for the implicit function that was denoted by $h(y)$ in the previous theorem, and $u^h_\beta(\beta)$ for the derivatives.

Theorem 5.2 (Local Parameterization of Solutions in R^n) *Suppose that we have a set of n algebraic state equations involving a discrete state variable $u^h \in R^n$ and a set of parameters $\beta \in R^m$, which we write*

$$F^h(u^h, \beta) = 0; \tag{5.6}$$

that these state equations are well-defined and continuously differentiable in both u^h and β, throughout some open domain $\mathcal{D} \subset R^n \times R^m$; that we have a pair of values u^h_0 and β_0, with $(u^h_0, \beta_0) \in \mathcal{D}$, for which $F^h(u^h_0, \beta_0) = 0$; and that the linear system of equations

$$F^h_u(u^h_0, \beta_0)\psi = \phi \tag{5.7}$$

is uniquely solvable for any value ϕ.

Then there exists an open set $U \subset R^n \times R^m$, containing (u^h_0, β_0), an open set $V \subset R^m$ containing β_0, and a function $u^h : V \to R^n$ such that $u^h(\beta_0) = u_0$ and $F^h(u^h(\beta), \beta) = 0$ for all $\beta \in V$.

Furthermore, the function $u^h(\beta)$ is uniquely determined by $(u^h(\beta), \beta) \in U$ for all $\beta \in V$, and $u^h(\beta) \in C^1(V)$, and the derivative function $u^h_\beta(\beta)$ satisfies

$$F^h_u(u^h(\beta), \beta)u^h_\beta(\beta) + F^h_\beta(u^h(\beta), \beta) = 0. \tag{5.8}$$

PROOF: Condition (5.7) implies that

$$rank(F^h_u(u^h_0, \beta_0)) = n, \tag{5.9}$$

and since all the other hypotheses of Theorem 5.2 are assumed, we may now assert the desired result. □

Thus, the implicit function theorem says that if we solve the discrete problem for a particular set of parameters β_0, getting a solution u_0^h, then for values of β "near" to β_0, there is a uniquely determined "nearby" state solution value, which we may write as $u^h(\beta)$. The pair $(u^h(\beta), \beta)$ satisfy the state equation, and $u^h(\beta)$ is continuously differentiable.

Since the discrete state equations are algebraic, F_u^h is simply the Jacobian matrix, and Condition (5.7) requires that this matrix be nonsingular. Given that, the derivative function $u_\beta^h(\beta)$ is defined by a solvable linear system.

Thus if we can assume the conditions of Theorem 5.2, then when we have a particular solution to the parameterized state equations, we know that there is actually a smoothly parameterized curve of solutions, which we may write as $u^h(\beta)$, of "nearby" solutions to "nearby" problems. We are interested in studying exactly how these nearby solutions vary with changes in the parameter. We make the following definitions:

Definition 5.1 (Discrete Sensitivity Equation) *Equation (5.8) is called the* **discrete sensitivity equation** *corresponding to the discrete state equation (5.6).*

Definition 5.2 (Discrete Sensitivity) *Under the assumptions of Theorem 5.2, the function $u_\beta^h(\beta)$ which solves the discrete sensitivity equation (5.8) is defined to be the* **discrete sensitivity** u_0^h *with respect to the parameters β.*

The expression $u_\beta^h(\beta)$ stands for the sensitivity of the discrete solution with respect to all of the parameters. For our problem, u^h actually represents three discretized functions, which are in turn represented by $2N_w + N_q$ real numbers. Supposing that we have $NPar$ parameters, then $u_\beta^h(\beta)$ could be expressed as a rectangular matrix of $2N_w + N_q$ rows and $NPar$ columns. A particular column would represent the sensitivity of all of the solution coefficients with respect to a particular parameter β_i, and we could denote this by $u_{\beta_i}^h(\beta)$, which is also equal to the partial derivative of $u^h(\beta)$ with respect to β_i.

Under the assumption that the state equations $F(u^h, \beta) = 0$ contain no derivatives of the state variables with respect to β, the sensitivity equation (5.8) is *linear* in the unknown function u^h_β.

Perhaps we should emphasize the meaning of Equation (5.8). It tells us that to compute the sensitivity of a particular solution to the state equations, we must differentiate the state equations with respect to the parameter of interest, and evaluate the resulting quantities at the particular solution, to arrive at the sensitivity equations.

5.5 The Discretized Sensitivity of a Solution

If we start with a continuous state equation for a state variable u, discretize it to a system for a discrete state variable u^h, we know from the previous section that we may also derive a sensitivity equation from this discretized state equation, for a quantity u^h_β, and that the sensitivity u^h_β is exactly the partial derivative of the discrete state variable u^h with respect to β.

Suppose, instead, that, beginning with the same continuous state equation, we first derive the continuous sensitivity equation, and *then* discretize that. We have interchanged the two operations that gave us the sensitivity u^h_β. What can we say about the results of this operation?

When we discretize the state equation first, we derive the sensitivity equation by differentiation of the discretized system. This can require the differentiation of numerous terms that do not appear directly in the state equation, but are simply part of the discretization process that may depend on the parameters, such as the location of quadrature points, the area of finite elements, the form of basis functions, and so on.

By contrast, it is generally easy to differentiate the continuous state equation with respect to a parameter. Once this is done, the same process that discretized the state equation can be applied to the sensitivity equation. This means that there is far less work required to design the algorithm, and we will see later that there will also be far less work to carry out the algorithm as well.

Before we proceed, we give a name to the solution of the system that is produced by discretizing the continuous sensitivity equation.

Definition 5.3 (Discretized Sensitivity of a Solution) *Suppose that we have a continuous state equation $F(u, \beta) = 0$ with solution u, and the corresponding continuous sensitivity equation with sensitivity u_β. Suppose that a discretization can be applied, yielding a discretized state equation $F^h(u^h, \beta) = 0$. Suppose that this same discretization can be applied to the continuous sensitivity equation (5.2), yielding what will be called the* **discretized sensitivity equation***:*

$$(F_u)^h(u^h(\beta), \beta)\ (u_\beta)^h + (F_\beta)^h(u^h(\beta), \beta) = 0\ . \tag{5.10}$$

The solution $(u_\beta)^h$ of this equation, if any, is called the **discretized sensitivity** *of the continuous solution u.*

Because a discretization operation has been applied, the discretized sensitivity equation is not guaranteed to be a sensitivity equation, that is, it need not represent the defining equation for a partial derivative.

Moreover, when we solve the discretized sensitivity equations, the discretized sensitivities are *not* guaranteed to actually be the partial derivatives of any quantity. In particular, they are not guaranteed to be the partial derivatives of the discrete state variables with respect to the parameters.

It is easy to miss this fact, because in many common cases, the operations of differentiation and discretization may commute. That is, we *may* have the case that:

$$(u_\beta)^h = u_\beta^h, \tag{5.11}$$

but such a situation must be verified.

In our setting, because of the convergence properties of the finite element method, we can assert that, as $h \to 0$ the discretized sensitivity $(u_\beta)^h$ approaches u_β, the sensitivity of the continuous solution. But even this statement doesn't guarantee commutativity, since we haven't shown that u_β^h also approaches u_β.

5.6 Example: An Explicit Parameter in a Linear ODE

We will now consider a simple case where the sensitivities are easy to compute. Consider the following linear ordinary differential equation, with parameter β, which will play the role of our continuous state equation:

$$\frac{du(t,\beta)}{dt} = \beta\, u(t,\beta), \tag{5.12}$$

$$u(0,\beta) = u_0. \tag{5.13}$$

For any particular value of β, the exact solution, that is, our continuous state solution, is:

$$u(t,\beta) = u_0\, e^{\beta t}, \tag{5.14}$$

from which we may immediately compute the continuous sensitivity:

$$u_\beta(t,\beta) = \frac{\partial u}{\partial \beta} = u_0\, t\, e^{\beta t}\ . \tag{5.15}$$

We now show that we do not need to have an explicit formula for the dependence of u on β in order to compute u_β. To show this, we compute the continuous sensitivity equations:

$$\frac{du_\beta(t,\beta)}{dt} = \beta\, u_\beta(t,\beta) + u(t,\beta), \tag{5.16}$$

$$u_\beta(0,\beta) \;=\; 0. \tag{5.17}$$

For a fixed value of β, we can solve this system directly for $u_\beta(t,\beta)$. We would not need a formula for u valid for all β, but only the form of $u(t,*)$ for the current values of the parameters.

Now we suppose that we wish to define and solve a discretized version of this problem. We discretize by setting down a mesh of equally spaced points t_i, with a spacing of h, and with the initial point set to $t_0 = 0$, and then using the forward Euler approximation for the derivative with respect to t. Then we may immediately write the resulting discrete state equations as:

$$\frac{u^h(t_{i+1},\beta) - u^h(t_i,\beta)}{h} \;=\; \beta u^h(t_i,\beta), \tag{5.18}$$

$$u^h(t_0,\beta) \;=\; u_0. \tag{5.19}$$

Now suppose that we have solved the discrete state equation for the state variables $u^h(t_i,\beta)$. We may then differentiate the discrete state equations with respect to β to get the sensitivity equations for the discretized problem:

$$\frac{u^h_\beta(t_{i+1},\beta) - u^h_\beta(t_i,\beta)}{h} \;=\; \beta u^h_\beta(t_i,\beta) + u^h(t_i,\beta), \tag{5.20}$$

$$u^h_\beta(t_0,\beta) \;=\; 0\ . \tag{5.21}$$

If we instead discretize the continuous sensitivity equation, we arrive at the discretized sensitivity equations:

$$\frac{(u_\beta)^h(t_{i+1},\beta) - (u_\beta)^h(t_i,\beta)}{h} \;=\; \beta (u_\beta)^h(t_i,\beta) + u^h(t_i,\beta), \tag{5.22}$$

$$(u_\beta)^h(t_0,\beta) \;=\; 0\ . \tag{5.23}$$

The equations defining u^h_β and $(u_\beta)^h$ are the same, and so their solutions must be the same. For this problem and this discretization, then, differentiation and discretization may be applied in either order without affecting the result.

5.7 Sensitivities for Geometric Parameters

We have seen that the derivation of the sensitivity equations can be quite straightforward when the parameter in question appears explicitly in the state equations. Simply differentiating the state equations produces the desired system.

But geometric parameters do not appear in such a simple way. For instance, a state equation involving a geometric parameter might have the form $u(\beta) = 0$, where β is to be interpreted here as a spatial location. Our simple technique of differentiation will not produce a proper result here.

To produce a proper formulation of the sensitivity with respect to a geometric parameter, it is best, if possible, to return to the definition of a sensitivity as a difference quotient. In other words, given a solution $u(x, \beta_0)$, we can compute its sensitivity to the parameter β by imagining that we compute a solution at $\beta_0 + \Delta\beta$, and considering the limit

$$u_\beta(x, \beta_0) = \lim_{\Delta\beta \to 0} \frac{u(x, \beta_0 + \Delta\beta) - u(x, \beta_0)}{\Delta\beta} \ . \tag{5.24}$$

The difference quotient makes a comparison between values of u at the same spatial coordinates. Because a geometric parameter can actually alter the shape of the region, we may find that even for small perturbations of the parameter, a particular point x no longer lies within the region, and has no associated value of u. This problem is particularly acute when the point x lies on the boundary of the region. However, there are generally ways to handle such difficulties by extending or extrapolating the solution as necessary. We will see, when we consider the sensitivities with respect to bump parameters, that for our formulation, we can always choose the sign of the perturbation of the bump parameter so that the point of interest remains in the flow region, and hence has defined solution values that may be used to make the necessary comparison.

It should be clear that geometric parameters are more difficult to handle when computing sensitivities. In the next chapter, we will see that geometric parameters can also create situations in which the discretization and differentiation operations do not commute.

Chapter 6

SENSITIVITIES FOR AN EXPLICIT PARAMETER

6.1 Introduction

We now consider the computation of sensitivities for a Navier Stokes problem. We are interested in both the continuous and discrete problems, and we would like an idea of the form of the resulting sensitivity equations, and perhaps an idea of what a sensitivity "looks like" for such a problem.

We consider how, starting with a general set of continuous Navier Stokes equations, we may derive the sensitivity equations with respect to the Reynolds number Re, or a parameter λ that controls the strength of the inflow. We derive the corresponding discrete sensitivity equations for the discretized Navier Stokes equations. We then discretize the continuous sensitivity equation, to derive the equation for the discretized sensitivities.

We make a simple check of these calculations for the Poiseuille flow solution, computing the continuous, discrete and discretized sensitivities directly, and plugging them into the derived equations.

We display pictures of some computations of sensitivities for a discrete problem involving fluid flow in a channel with a bump. We also exhibit a table showing excellent agreement between the discrete sensitivities and the discretized sensitivities.

6.2 *Re*-Sensitivity Equations

Let us suppose that our parameter is the Reynolds number, Re, and that for a particular value of Re_0, we have a state solution (u_0, v_0, p_0). We assume that there is an open neighborhood of Re_0, so that, for any Re in that neighborhood, and for any (x, y) in the flow region, the mixed derivatives of the state variables with respect to Re and either x or y exist and are continuous. Then we can apply Clairaut's theorem, which allows us to interchange the order of differentiations with respect to Re and either x or y.

The Reynolds number appears explicitly, and in a simple way, in the horizontal and vertical momentum equations. We differentiate the continuous state equations with respect to Re, and where necessary, interchange so that differentiation of state variables with respect to Re happens first. We then abbreviate the expression $\dfrac{\partial u}{\partial Re}$ as u_{Re}.

With these assumptions, we arrive at the following set of *continuous Re-sensitivity equations*:

$$\begin{aligned}
-(\frac{\partial^2 u_{Re}}{\partial x^2} + \frac{\partial^2 u_{Re}}{\partial y^2}) \;&+\; Re(u_{Re}\frac{\partial u}{\partial x} + u\frac{\partial u_{Re}}{\partial x} + v_{Re}\frac{\partial u}{\partial y} + v\frac{\partial u_{Re}}{\partial y} + \frac{\partial p_{Re}}{\partial x}) \\
&= -(u\frac{\partial u}{\partial x} + v\frac{\partial u}{\partial y} + \frac{\partial p}{\partial x}) \qquad (6.1) \\
-(\frac{\partial^2 v_{Re}}{\partial x^2} + \frac{\partial^2 v_{Re}}{\partial y^2}) \;&+\; Re(u_{Re}\frac{\partial v}{\partial x} + u\frac{\partial v_{Re}}{\partial x} + v_{Re}\frac{\partial v}{\partial y} + v\frac{\partial v_{Re}}{\partial y} + \frac{\partial p_{Re}}{\partial y}) \\
&= -(u\frac{\partial v}{\partial x} + v\frac{\partial v}{\partial y} + \frac{\partial p}{\partial y}) \qquad (6.2) \\
\frac{\partial u_{Re}}{\partial x} + \frac{\partial v_{Re}}{\partial y} &= 0 \qquad (6.3)
\end{aligned}$$

with differentiated boundary conditions:

$$v_{Re}(x, y) = 0 \text{ along the boundary;}$$

$$\begin{aligned}
u_{Re}(x,y) &= 0 \text{ along the inflow boundary, the walls, and the bump;}\\
\frac{\partial u_{Re}}{\partial x}(xmax,y) &= 0 \text{ on the outflow;}\\
p_{Re}(xmax,ymax) &= 0.
\end{aligned}$$

To get the discrete *Re*-sensitivity equations, we recall the discrete state equations:

$$\int_\Omega (\frac{\partial u^h}{\partial x}\frac{\partial w_i}{\partial x} + \frac{\partial u^h}{\partial y}\frac{\partial w_i}{\partial y} \quad + \quad Re(u^h\frac{\partial u^h}{\partial x} + v^h\frac{\partial u^h}{\partial y} + \frac{\partial p^h}{\partial x})w_i)\, d\Omega = 0 \tag{6.4}$$

$$\int_\Omega (\frac{\partial v^h}{\partial x}\frac{\partial w_i}{\partial x} + \frac{\partial v^h}{\partial y}\frac{\partial w_i}{\partial y} \quad + \quad Re(u^h\frac{\partial v^h}{\partial x} + v^h\frac{\partial v^h}{\partial y} + \frac{\partial p^h}{\partial y})w_i)\, d\Omega = 0 \tag{6.5}$$

$$\int_\Omega (\frac{\partial u^h}{\partial x} + \frac{\partial v^h}{\partial y})q_i\, d\Omega \quad = \quad 0 \tag{6.6}$$

and boundary conditions:

$$v^h(x,y) = 0 \text{ at boundary velocity nodes;} \tag{6.7}$$

$$u^h(0,y) = Inflow^h(y,\lambda) \text{ at inflow velocity nodes;} \tag{6.8}$$

$$u^h(x,y) = 0 \text{ at velocity nodes on the walls and the bump;} \tag{6.9}$$

$$p^h(xmax,ymax) = 0 \text{ for the upper right pressure node.} \tag{6.10}$$

If we now differentiate these equations with respect to *Re*, bringing the differentiation inside the integral signs and interchanging differentiations where necessary, we arrive at the *discrete Re-sensitivity equations*:

$$\begin{aligned}
\int_\Omega (\frac{\partial u^h_{Re}}{\partial x}\frac{\partial w_i}{\partial x} + \frac{\partial u^h_{Re}}{\partial y}\frac{\partial w_i}{\partial y} \quad &+ \quad Re(u^h_{Re}\frac{\partial u^h}{\partial x} + u^h\frac{\partial u^h_{Re}}{\partial x} + v^h_{Re}\frac{\partial u^h}{\partial y} + v^h\frac{\partial u^h_{Re}}{\partial y} + \frac{\partial p^h_{Re}}{\partial x})w_i)\, d\Omega \\
&= \quad \int_\Omega -(u^h\frac{\partial u^h}{\partial x} + v^h\frac{\partial u^h}{\partial y} + \frac{\partial p^h}{\partial x})w_i\, d\Omega
\end{aligned} \tag{6.11}$$

$$\begin{aligned}
\int_\Omega (\frac{\partial v^h_{Re}}{\partial x}\frac{\partial w_i}{\partial x} + \frac{\partial v^h_{Re}}{\partial y}\frac{\partial w_i}{\partial y} \quad &+ \quad Re(u^h_{Re}\frac{\partial v^h}{\partial x} + u^h\frac{\partial v^h_{Re}}{\partial x} + v^h_{Re}\frac{\partial v^h}{\partial y} + v^h\frac{\partial v^h_{Re}}{\partial y} + \frac{\partial p^h_{Re}}{\partial y})w_i)\, d\Omega \\
&= \quad \int_\Omega -(u^h\frac{\partial v^h}{\partial x} + v^h\frac{\partial v^h}{\partial y} + \frac{\partial p^h}{\partial y})w_i\, d\Omega
\end{aligned} \tag{6.12}$$

$$\int_\Omega (\frac{\partial u^h_{Re}}{\partial x} + \frac{\partial v^h_{Re}}{\partial y}) q_i \, d\Omega \;=\; 0 \qquad (6.13)$$

with boundary conditions:

$$\begin{aligned}
v^h_{Re}(x,y) &= 0 \text{ for boundary velocity nodes;} \\
u^h_{Re}(x,y) &= 0 \text{ for velocity nodes on the inflow, walls, and bump;} \\
p^h_{Re}(xmax, ymax) &= 0 \text{ at the upper right pressure node.}
\end{aligned}$$

On the other hand, if we apply the finite element method to the sensitivity equations for the continuous system, we get the equations for the discretized *Re*-sensitivities:

$$\begin{aligned}
\int_\Omega (\frac{\partial (u_{Re})^h}{\partial x}\frac{\partial w_i}{\partial x} + \frac{\partial (u_{Re})^h}{\partial y}\frac{\partial w_i}{\partial y} &+ Re((u_{Re})^h \frac{\partial u^h}{\partial x} + u^h \frac{\partial (u_{Re})^h}{\partial x} + (v_{Re})^h \frac{\partial u^h}{\partial y} + v^h \frac{\partial (u_{Re})^h}{\partial y} + \frac{\partial (p_{Re})^h}{\partial x}) w_i) \; c \\
&= \int_\Omega -(u^h \frac{\partial u^h}{\partial x} + v^h \frac{\partial u^h}{\partial y} + \frac{\partial p^h}{\partial x}) w_i \, d\Omega \qquad (6.1 \\
\int_\Omega (\frac{\partial (v_{Re})^h}{\partial x}\frac{\partial w_i}{\partial x} + \frac{\partial (v_{Re})^h}{\partial y}\frac{\partial w_i}{\partial y} &+ Re((u_{Re})^h \frac{\partial v^h}{\partial x} + u^h \frac{\partial (v_{Re})^h}{\partial x} + (v_{Re})^h \frac{\partial v^h}{\partial y} + v^h \frac{\partial (v_{Re})^h}{\partial y} + \frac{\partial (p_{Re})^h}{\partial y}) w_i) \; d \\
&= \int_\Omega -(u^h \frac{\partial v^h}{\partial x} + v^h \frac{\partial v^h}{\partial y} + \frac{\partial p^h}{\partial y}) w_i \, d\Omega \qquad (6.1 \\
\int_\Omega (\frac{\partial (u_{Re})^h}{\partial x} + \frac{\partial (v_{Re})^h}{\partial y}) q_i \, d\Omega &= 0 \qquad (6.1
\end{aligned}$$

with boundary conditions:

$$\begin{aligned}
(v_{Re})^h(x,y) &= 0 \text{ for boundary velocity nodes;} \\
(u_{Re})^h(x,y) &= 0 \text{ for velocity nodes on the inflow, walls, and bump;} \\
(p_{Re})^h(xmax, ymax) &= 0 \text{ at the upper right pressure node.}
\end{aligned}$$

A glance at the discrete sensitivity equations and the discretized sensitivity equations shows that we have arrived at the same system; hence, for the *Re* parameter, the discrete and discretized sensitivities are identical.

6.3 λ-Sensitivity Equations

We now derive the sensitivity equations for λ, a parameter that influences the strength of the $Inflow$ function. Exactly as for *Re*, we proceed to derive the *continuous λ-sensitivity equations*:

$$\begin{aligned}
-(\frac{\partial^2 u_\lambda}{\partial x^2} + \frac{\partial^2 u_\lambda}{\partial y^2}) + Re(u_\lambda \frac{\partial u}{\partial x} + u\frac{\partial u_\lambda}{\partial x} + v_\lambda \frac{\partial u}{\partial y} + v\frac{\partial u_\lambda}{\partial y} + \frac{\partial p_\lambda}{\partial x}) &= 0 \qquad (6.17)\\
-(\frac{\partial^2 v_\lambda}{\partial x^2} + \frac{\partial^2 v_\lambda}{\partial y^2}) + Re(u_\lambda \frac{\partial v}{\partial x} + u\frac{\partial v_\lambda}{\partial x} + v_\lambda \frac{\partial v}{\partial y} + v\frac{\partial v_\lambda}{\partial y} + \frac{\partial p_\lambda}{\partial y}) &= 0 \qquad (6.18)\\
\frac{\partial u_\lambda}{\partial x} + \frac{\partial v_\lambda}{\partial y} &= 0 \qquad (6.19)
\end{aligned}$$

with differentiated boundary conditions:

$$\begin{aligned}
v_\lambda(x,y) &= 0 \text{ along the boundary;}\\
u_\lambda(x,y) &= \frac{\partial Inflow(y,\lambda)}{\partial \lambda} \text{ along the inflow boundary;}\\
u_\lambda(x,y) &= 0 \text{ along the walls, and the bump;}\\
\frac{\partial u_\lambda}{\partial x}(xmax,y) &= 0 \text{ on the outflow;}\\
p_\lambda(xmax,ymax) &= 0.
\end{aligned}$$

As for the *Re* parameter, we are about to discover that the discrete sensitivity equations and the discretized sensitivity equations are identical. Therefore, we will only write out one set of equations.

Applying the finite element method to the continuous sensitivity equations, we get the *discretized λ-sensitivity equation*:

$$\begin{aligned}
\int_\Omega (\frac{\partial (u_\lambda)^h}{\partial x}\frac{\partial w_i}{\partial x} &+ \frac{\partial (u_\lambda)^h}{\partial y}\frac{\partial w_i}{\partial y}\\
&+ Re((u_\lambda)^h \frac{\partial u^h}{\partial x} + u^h \frac{\partial (u_\lambda)^h}{\partial x} + (v_\lambda)^h \frac{\partial u^h}{\partial y} + v^h \frac{\partial (u_\lambda)^h}{\partial y} + \frac{\partial (p_\lambda)^h)}{\partial x})w_i)\ d\Omega = 0 \qquad (6.20)\\
\int_\Omega (\frac{\partial (v_\lambda)^h}{\partial x}\frac{\partial w_i}{\partial x} &+ \frac{\partial (v_\lambda)^h}{\partial y}\frac{\partial w_i}{\partial y}
\end{aligned}$$

$$+ \quad Re((u_\lambda)^h \frac{\partial v^h}{\partial x} + u^h \frac{\partial (v_\lambda)^h}{\partial x} + (v_\lambda)^h \frac{\partial v^h}{\partial y} + v^h \frac{\partial (v_\lambda)^h}{\partial y} + \frac{\partial (p_\lambda)^h)}{\partial y}) w_i)\ d\Omega = 0 \tag{6.21}$$

$$\int_\Omega (\frac{\partial (u_\lambda)^h}{\partial x} \quad + \quad \frac{\partial (v_\lambda)^h}{\partial y}) q_i\ d\Omega = 0 \tag{6.22}$$

with boundary conditions:

$$\begin{aligned}
(v_\lambda)^h(x,y) &= 0 \text{ for boundary velocity nodes;} \\
(u_\lambda)^h(0,y) &= \frac{\partial Inflow^h(y,\lambda)}{\partial \lambda} \text{ for outflow velocity nodes;} \\
(u_\lambda)^h(x,y) &= 0 \text{ for velocity nodes on the walls and bump;} \\
(p_\lambda)^h)(xmax, ymax) &= 0 \text{ for the upper right pressure node.}
\end{aligned}$$

Again, because the discrete sensitivity equations are identical in form to the discretized sensitivity equations, we may assert the equality of the discrete sensitivities u_λ^h and the discretized sensitivities $(u_\lambda)^h$.

6.4 Discretization and Differentiation May Commute

For both the Reynolds and inflow parameters, we reached the same set of equations by two distinct methods, starting from the continous state equations:

- Discretize to determine the discrete state equations, then differentiate to determine the discrete sensitivity equations;
- Differentiate to determine the continuous sensitivity equations, then discretize to determine the discretized sensitivity equations.

The commutation of these two operations, if it occurs, is quite convenient. It allows us to do the sensitivity operations in a continuous space, where they are easy to perform, and to apply the discretization last, which means we can rely on a variety of theorems for the finite element method to show convergence of the discretized sensitivities to the continuous sensitivities. Thus, if we have commutation, then at the same time, the discretized sensitivities are the exact partial derivatives of the discrete solution with respect to the parameter (useful for various calculations), *and* there are the finite element approximants to the continuous sensitivities (which is useful for convergence analysis).

In fact, we can show that this commutation occurs between the application of the finite element method and differentiation with respect to parameters, as long as the parameters don't influence the region, the basis functions, or other items that would affect the finite element implementation:

Theorem 6.1 (Commutation for Non-Geometric Parameters) *Consider a system of differential equations $F(u, \beta) = 0$ with boundary conditions $G(u, \beta) = 0$ for the unknown function $u(x, y)$. Suppose that this system is suitable for treatment by the finite element method. Suppose that the implicit function theorem may be applied, so that $u(x, y)$ may be regarded as a continuously differentiable function of β, which we write $u(x, y, \beta)$. Suppose that F and G are continuously differentiable in their arguments u and β. Suppose that the operations of differentiation and discretization commute on both F and G, so that we may write*

$$F_u^h = (F^h)_u = (F_u)^h \tag{6.23}$$

$$F_\beta^h = (F^h)_\beta = (F_\beta)^h \tag{6.24}$$

$$G_u^h = (G^h)_u = (G_u)^h \tag{6.25}$$

$$G_\beta^h = (G^h)_\beta = (G_\beta)^h \tag{6.26}$$

Suppose, finally, that the parameter β has no influence on the shape of the domain Ω, the value of basis functions, or on other such features of the finite element discretization. Then the discretized sensitivity equations are identical to the sensitivity equations for the discretized *system.*

Proof: The continuous sensitivity equations will have the form:

$$F_u(u,\beta)u_\beta + F_\beta(u,\beta) = 0 \tag{6.27}$$

$$G_u(u,\beta)u_\beta + G_\beta(u,\beta) = 0 \tag{6.28}$$

and the discretized sensitivity equations will have the form:

$$\int_\Omega (F_u^h(u^h,\beta)(u_\beta)^h + F_\beta^h(u^h,\beta))\ w_i\ d\Omega = 0 \tag{6.29}$$

$$G_u^h(u^h,\beta)(u_\beta)^h + G_\beta^h(u^h,\beta) = 0 \tag{6.30}$$

where w_i is a "typical" basis function. The discretized state equations have the form:

$$\int_\Omega F^h(u^h,\beta)\ w_i\ d\Omega = 0 \tag{6.31}$$

$$G^h(u^h,\beta) = 0 \tag{6.32}$$

and the discrete sensitivity equations may be computed by carrying out the operations:

$$\frac{D}{D\beta}\int_\Omega F^h(u^h,\beta)\ w_i\ d\Omega = 0 \tag{6.33}$$

$$\frac{D}{D\beta}G^h(u^h,\beta) = 0 \tag{6.34}$$

Now because the domain of integration does not depend on β, we may bring the differentiation under the integral sign. Because the basis function w_i does not depend in any way on β, we have only to compute the total derivative of F^h, so that the discrete sensitivity equations are:

$$\int_\Omega (F_u^h(u^h,\beta)u_\beta^h + F_\beta^h(u^h,\beta))\ w_i\ d\Omega = 0 \tag{6.35}$$

$$G_u^h(u^h,\beta)u_\beta^h + G_\beta^h(u^h,\beta) = 0 \tag{6.36}$$

But this system is identical in form to the discretized sensitivity equation, and hence their solutions must also be identical. □

6.5 Exact Example: Poiseuille Sensitivities

For the Poiseuille flow problem, consider the influence of λ, the strength of the inflow field, upon the solution defined by Equations (3.6)-(3.8). The continuous sensitivities may be immediately written down from the explicit formula for the solution:

$$u_\lambda = \frac{\partial u(x,y)}{\partial \lambda} = y(ymax - y) \tag{6.37}$$

$$v_\lambda = \frac{\partial v(x,y)}{\partial \lambda} = 0 \tag{6.38}$$

$$p_\lambda = \frac{\partial p(x,y)}{\partial \lambda} = 2(xmax - x)/Re \tag{6.39}$$

We can make some simple conclusions from this information. For instance, λ has no influence on the vertical velocity anywhere. That's obvious, actually, since, for every value of λ the vertical velocity is zero everywhere. Note also how the influence of λ on the horizontal velocity varies: it is zero along the walls (where u is always zero) and is greatest at the center line $y = ymax/2$, because changes in the velocity are proportional to the value of the velocity, and the velocity is greatest along the center line.

The Reynolds number Re might also be considered a parameter for this problem, but would make a singularly uninteresting flow:

$$u_{Re} = 0 \tag{6.40}$$

$$v_{Re} = 0 \tag{6.41}$$

$$p_{Re} = -2\lambda(xmax - x)/Re^2 \tag{6.42}$$

The continuous sensitivity equations derived earlier may be checked by plugging in these continuous sensitivity functions. For the Re parameter, note that the vertical momentum and continuity equations are identically zero, while the horizontal momentum equation reduces

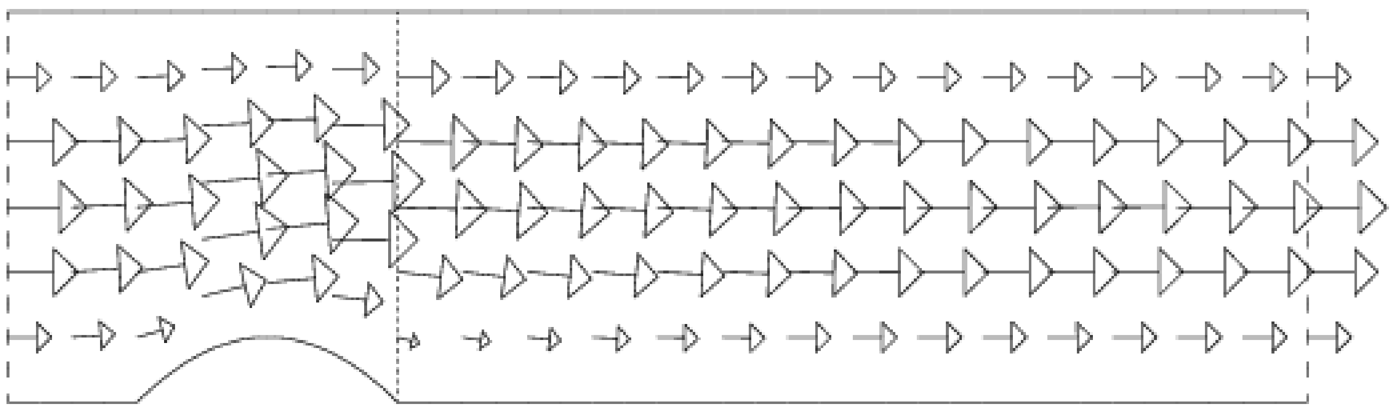

Figure 6.1: Discrete velocity sensitivity with respect to the inflow parameter λ.

to:

$$Re\frac{\partial p_{Re}}{\partial x} = -\frac{\partial p}{\partial x}, \tag{6.43}$$

which is easily verified by plugging in the values of p and p_{Re}.

For the λ parameter, the system simplifies to

$$\begin{aligned} -\frac{\partial^2 u_\lambda}{\partial y^2} + Re\frac{\partial p_\lambda}{\partial x} &= 0 \qquad (6.44)\\ \frac{\partial u_\lambda}{\partial x} &= 0 \qquad (6.45)\end{aligned}$$

which is again easy to verify.

6.6 Computational Example: Flow Past a Bump

In this section, we display plots of the discrete sensitivities of the velocity for a flow problem with parameters (λ, α, Re). The base solution was calculated at parameter values $(0.25, 0.25, 5.0)$. The underlying grid had a mesh parameter of $h = 0.25$.

For clarity, we have only shown the sensitivities at every other node. Also, the vectors have

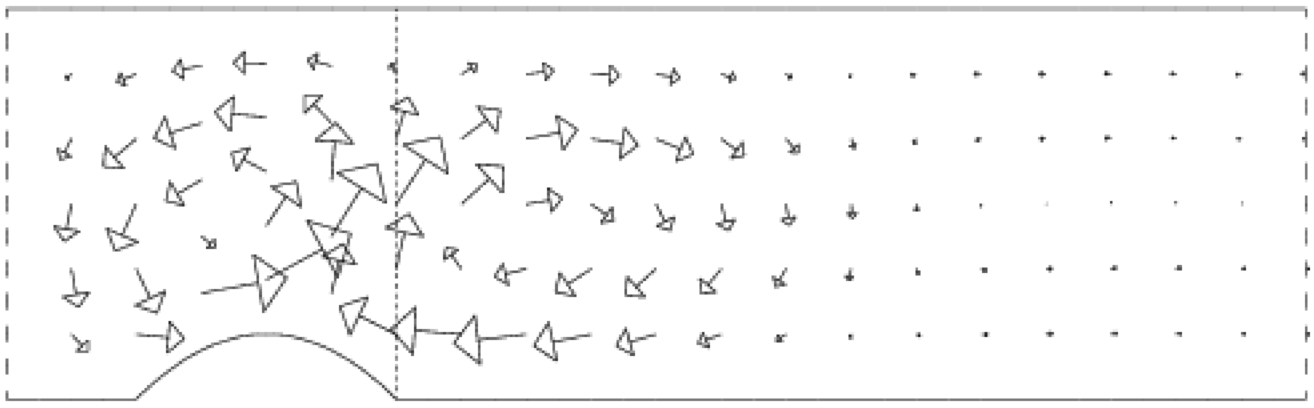

Figure 6.2: Discrete velocity sensitivity with respect to the *Re* parameter.

been resized so that the largest vector has a fixed size; thus, comparisons of the strength of the influence of different parameters is not possible. However, in any particular plot, the relative sizes of the vectors are meaningful, and represent the strength of the influence of the parameter on each particular velocity. A similar, but less enlightening, contour plot could be shown for pressure.

In the resulting plots, keep in mind that what we are displaying is a velocity *increment* that should be added to the current velocity if the given parameter is increased. The discrete velocity sensitivities for the inflow and *Re* parameters satisfy the discrete continuity equation, a fact suggested by the form of the displayed solutions in Figure 6.1 and Figure 6.2. Moreover, as can be judged from the plots, the λ-sensitivity equations have an inflow boundary term, while the *Re*-sensitivity equations have zero boundary conditions, but have source terms throughout the region. Note in Figure 6.1 how the inflow parameter affects the velocity field throughout the entire region, while Figure 6.2 shows that the influence of the *Re* parameter is restricted to the area near the bump.

Table 6.1: Finite difference check of λ-sensitivities.

Variable	$\|u^h_\lambda\|_\infty$	$\|\Delta(u^h)/\Delta(\lambda)\|_\infty$	$\|\text{Difference}\|_\infty$
U	1.167	1.167	1.1E-08
V	0.2067	0.2067	6.7E-09
P	1.140	1.140	5.6E-08

Table 6.2: Finite difference check of Re-sensitivities.

Variable	$\|u^h_{Re}\|_\infty$	$\|\Delta(u^h)/\Delta(Re)\|_\infty$	$\|\text{Difference}\|_\infty$
U	3.118E-03	3.118E-03	2.1E-09
V	2.460E-03	2.460E-03	1.3E-09
P	5.334E-02	5.334E-02	5.2E-08

6.7 Comparison of Sensitivities with Finite Difference Estimates

We have mentioned that the discrete sensitivities are an inexpensive estimate of the partial derivatives of the discrete state variables with respect to a parameter. We have also noted that the accuracy of this approximation is limited, and depends on the fineness of the mesh parameter h.

To illustrate that the sensitivities can, in fact, provide a very good approximation in practice, Tables 6.1 and 6.2 compare the sensitivities with an estimate made using finite differences. The problem involves three parameters, (λ, α, Re), and our solution was computed at $(0.5, 0.5, 10.0)$, on a grid with $h = 0.125$. Here we will only look at the results for Re and λ. These results show almost perfect agreement between the two calculations.

The formula for $\Delta(\lambda)$ requires the value of ϵ, the relative accuracy for double precision arithmetic on the given computer, which, for the DEC Alpha, is roughly 1.1E-16:

$$\Delta(\lambda) = 10\epsilon^{\frac{1}{2}}\ sign(\lambda)\ (\|\lambda\| + 1), \tag{6.46}$$

which results in a relative perturbation of about 1.0E-07. The same sort of formula is used to compute $\Delta(Re)$.

Chapter 7

SENSITIVITIES FOR AN IMPLICIT PARAMETER

7.1 Introduction

In Chapter 6, we found that we could derive the sensitivity equations for an explicit variable by simple differentiation of the state equations. But if our parameter is *implicit*, that is, does not show up directly in the state equations, then simply formally differentiating the state equations will produce a homogeneous sensitivity system with zero solution, which can't be correct if the parameter does actually have some influence on the solution.

To derive the correct approach for computing such sensitivities, we need to return to the definition of a sensitivity as a change in the solution caused by changes in the parameter. In our examples, the influence of the parameter will show up in a nonzero source term in the boundary conditions.

The implicit parameters we will consider affect the geometry of the region. This means that we may have to make special adjustments when computing a partial derivative or making a finite difference quotient, at a fixed point in space. Special difficulties occur for points on the boundary, since even for small perturbations of the parameter, such a point is likely to move to the interior or exterior of the region. Moreover, the very structure of the discretization method will change, including the location of nodes, the shape of elements, and the definition of basis functions.

7.2 The Continuous α-Sensitivity Equations

We consider our standard problem of fluid flow in a channel with a bump, and attempt to derive the continuous sensitivity equations for a parameter that affects the shape of that bump. As usual, we denote this shape parameter by α, and note that while there might actually be several shape parameters in a particular problem, we will restrict our attention to a single one.

Let us suppose, then, that we have a particular parameter value α_0 and a solution $(u, v, p)(\alpha_0)$ to the corresponding flow problem. We will first consider the case of a point (x, y) lying in the interior of the flow region. We will do so by computing a second flow solution at a slightly altered parameter value, $\alpha_0 + \Delta\alpha$ and comparing the two solutions at (x, y). As long as the influence of the parameter α on the shape of the region is at least continuous, then because the point is contained strictly in the interior of the region, for all sufficiently small values of $\Delta\alpha$, the point will remain in the interior. Therefore, for both the unperturbed and perturbed parameter values, there will be a flow solution defined at the point. Hence, at such a point (x, y), we can define difference quotients of the flow solutions, and hence functions u_α, v_α and p_α which are the limits of those difference quotients. That is:

$$u_\alpha(x, y) \equiv \lim_{\Delta\alpha \to 0} \frac{u(\alpha_0 + \Delta\alpha) - u(\alpha_0)}{\Delta\alpha}. \tag{7.1}$$

These limit functions, the continuous sensitivities, will satisfy the homogeneous Oseen equations:

$$-(\frac{\partial^2 u_\alpha}{\partial x^2} + \frac{\partial^2 u_\alpha}{\partial y^2}) + Re\ (u_\alpha \frac{\partial u}{\partial x} + u \frac{\partial u_\alpha}{\partial x} + v_\alpha \frac{\partial u}{\partial y} + v \frac{\partial u_\alpha}{\partial y} + \frac{\partial p_\alpha}{\partial x}) = 0 \tag{7.2}$$

$$-(\frac{\partial^2 v_\alpha}{\partial x^2} + \frac{\partial^2 v_\alpha}{\partial y^2}) + Re\ (u_\alpha \frac{\partial v}{\partial x} + u \frac{\partial v_\alpha}{\partial x} + v_\alpha \frac{\partial v}{\partial y} + v \frac{\partial v_\alpha}{\partial y} + \frac{\partial p_\alpha}{\partial y}) = 0 \tag{7.3}$$

$$\frac{\partial u_\alpha}{\partial x} + \frac{\partial v_\alpha}{\partial y} = 0 \tag{7.4}$$

To complete the specification of our continuous sensitivity system, we have only to determine the form of the boundary conditions.

The pressure condition is easily disposed of. For every value of α, we require that $p(xmax, ymax) = 0$. Note that the values of $xmax$ and $ymax$ do not depend on the parameter; hence there is little doubt about how to take a difference quotient of this condition, and the limit operation results in the condition:

$$p_\alpha(xmax, ymax) = 0. \tag{7.5}$$

Now let us consider a point on a stationary wall. Both the horizontal and vertical velocities are zero here, for every value of α, and no pressure condition is applied. Hence the parameter can exert no influence whatsoever, and the appropriate boundary conditions are

$$u_\alpha = v_\alpha = 0 \text{ along the fixed walls.} \tag{7.6}$$

For similar reasons, we also easily determine the boundary condition:

$$\frac{\partial u_\alpha}{\partial x} = v_\alpha = 0 \text{ on the outflow boundary.} \tag{7.7}$$

The fact that the inflow velocity specification is always the same, regardless of the value of α means that we can also conclude that

$$u_\alpha = v_\alpha = 0 \text{ on the inflow boundary.} \tag{7.8}$$

At this point, we have almost specified a homogeneous problem. If the boundary condition along the bump were zero, then the solution to the Oseen problem would be the zero flow. Since apparently, this boundary condition will be the only way for the solution to be nonzero, the correct computation and accurate evaluation of this condition is crucial.

Recall that we have a function that defines the surface of the bump, of the form $y = Bump(x, \alpha)$. We now consider a fixed point (x, y) which, for the value $\alpha = \alpha_0$ lies on the bump surface. We want to analyze how the horizontal velocity at that point must be affected by changes in α. That means we must look at the value of $u(x, y)$ for solutions at nearby values of α. Of course, for any nearby value of α, the point (x, y) is probably no longer lying on the bump surface, and instead will probably lie outside the region, where we have no solution information, or in the interior, where we do. We much prefer the latter case. It seems reasonable to demand that our *Bump* function be specified so that, if a point (x, y) lies on the bump surface for the parameter value α_0, then it either remains on the bump surface for all values of α (the endpoints of the bump), or else, we can always find a perturbation $\Delta\alpha$ such that (x, y) lies strictly within $\Omega(\alpha)$ for every α strictly between α_0 and $\alpha_0 + \Delta\alpha$. Here, the perturbation $\Delta\alpha$ may in fact be negative.

Now we must try to compute an estimate for the value $u(x, y, \alpha_0 + \Delta\alpha)$ so that we can make our comparison. Since (x, y) was on the bump surface for $\alpha = \alpha_0$, and we have only changed α a small amount, it is reasonable to try to estimate the value of u at (x, y) by referring to its value at the nearby point $(x, Bump(x, \alpha_0 + \Delta\alpha))$. Since this reference point lies on the bump, we know that the value of u there is exactly zero:

$$u(x, Bump(x, \alpha_0 + \Delta\alpha), \alpha_0 + \Delta\alpha) = 0. \tag{7.9}$$

Using this information, we can use a Taylor estimate to deduce the value of u at (x, y):

$$u(x, y, \alpha_0 + \Delta\alpha) \quad = \quad u(x, Bump(x, \alpha_0 + \Delta\alpha), \alpha_0 + \Delta\alpha) \tag{7.10}$$

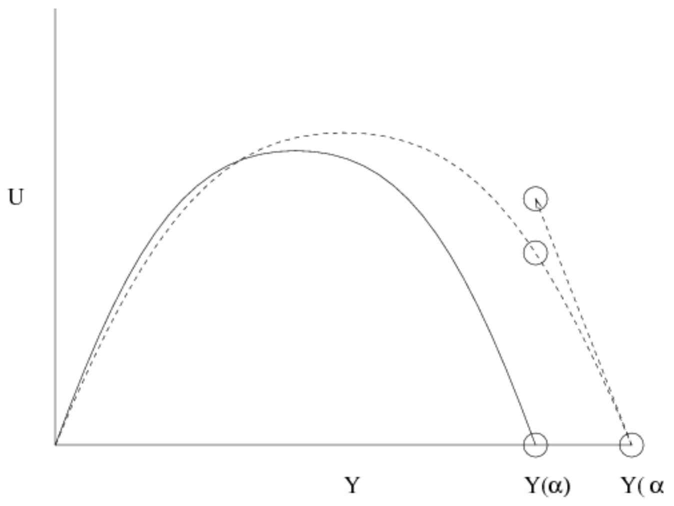

Figure 7.1: Estimating the solution at a moving point.
The original solution $u(x, y, \alpha_0)$ is the solid curve.
The perturbed solution $u(x, y, \alpha_0 + \Delta\alpha)$ is the dashed curve.
A two-term Taylor estimate approximates $u(x, \alpha_0, \alpha_0 + \Delta\alpha)$.

$$+ \quad \frac{\partial u}{\partial y}(x, \xi, \alpha_0 + \Delta\alpha)\ (y - Bump(x, \alpha_0 + \Delta\alpha)), \tag{7.11}$$

where ξ lies between $Bump(x, \alpha_0 + \Delta\alpha)$ and y. Using Equation (7.9) and the fact that $y = Bump(x, \alpha_0)$, our equation becomes:

$$u(x, y, \alpha_0 + \Delta\alpha) = \frac{\partial u}{\partial y}(x, \xi, \alpha_0 + \Delta\alpha)\ (Bump(x, \alpha_0) - Bump(x, \alpha_0 + \Delta\alpha)), \tag{7.12}$$

and if we assume that u is, at least locally, twice continuously differentiable in y, we can make an estimate using known quantities, with an error term:

$$\begin{aligned} u(x, y, \alpha_0 + \Delta\alpha) = &\frac{\partial u}{\partial y}(x, y, \alpha_0 + \Delta\alpha)\ (Bump(x, \alpha_0) - Bump(x, \alpha_0 + \Delta\alpha)) \\ &+O((Bump(x, \alpha_0) - Bump(x, \alpha_0 + \Delta\alpha))^2). \end{aligned} \tag{7.13}$$

Figure 7.1 suggests how we are going to estimate the value of u_α.

We now set up our difference quotient estimate, using the fact that $u(x, y, \alpha_0) = 0$:

$$u_\alpha(x, y, \alpha_0) \quad = \quad \frac{\partial u}{\partial \alpha}(x, y, \alpha_0) \tag{7.14}$$

$$\begin{aligned}
&= \lim_{\Delta\alpha\to 0} \frac{u(x,y,\alpha_0+\Delta\alpha)-u(x,y,\alpha_0)}{\Delta\alpha} && (7.15)\\
&= \lim_{\Delta\alpha\to 0} \frac{\frac{\partial u}{\partial y}(x,y,\alpha_0+\Delta\alpha)(Bump(x,\alpha_0)-Bump(x,\alpha_0+\Delta\alpha)}{\Delta\alpha} && (7.16)\\
&= -\frac{\partial u}{\partial y}(x,y,\alpha_0)\frac{\partial Bump}{\partial \alpha}(x,\alpha_0). && (7.17)
\end{aligned}$$

A second, and quicker, way to derive this same formula works by considering a point that *moves* with the surface of the bump. Such a point would have a varying y coordinate $y(\alpha) = Bump(x, \alpha)$. Then, for every value of α, the point lies on the bump and so the value of its horizontal velocity is zero. Therefore, we may assert:

$$\frac{Du}{D\alpha}(x, Bump(x,\alpha),\alpha) = \frac{\partial u}{\partial y}\frac{\partial Bump(x,\alpha)}{\partial \alpha} + \frac{\partial u}{\partial \alpha} = 0, \qquad (7.18)$$

which immediately yields Equation (7.17).

Similar manipulations allow us to conclude that

$$v_\alpha(x, Bump(x,\alpha),\alpha) = -\frac{\partial v}{\partial y}\frac{Bump(x,\alpha)}{\partial \alpha}. \qquad (7.19)$$

These two equations may be regarded as boundary conditions that are to replace the usual conditions on u and v on the bump. Our continuous α-sensitivity boundary conditions therefore have the form:

$$\begin{aligned}
u_\alpha(0,y) &= 0 \text{ on the inflow and walls;}\\
v_\alpha(0,y) &= 0 \text{ on the inflow, outflow and walls;}\\
u_\alpha(x,y) &= -\frac{\partial u}{\partial y}\frac{\partial Bump(x,\alpha)}{\partial \alpha} \text{ on the bump;}\\
v_\alpha(x,y) &= -\frac{\partial v}{\partial y}\frac{\partial Bump(x,\alpha)}{\partial \alpha} \text{ on the bump;}\\
\frac{\partial u_\alpha}{\partial x}(xmax,y) &= 0;\\
p_\alpha(xmax,ymax) &= 0.
\end{aligned}$$

These boundary conditions, together with the homogeneous Oseen equations, make up our continuous sensitivity system for the implicit parameter α.

7.3 The Discrete α-Sensitivity Equations

To derive the discrete α-sensitivity equations, we begin by writing out the discrete state equations (4.10), (4.11), and (4.12). We intend to differentiate these with respect to the shape parameter α. However, we cannot proceed by simply bringing the differentiation operator inside the integral; the case of α is special, because *the domain of integration depends on α.* We will try to emphasize that point in this section by dutifully designating the region as $\Omega(\alpha)$.

Thus, instead of the general rule:

$$\frac{\partial}{\partial\beta}\int_{\Omega} f(x,y,\beta)\;dxdy = \int_{\Omega}\frac{\partial f(x,y,\beta)}{\partial\beta}\;dxdy, \tag{7.20}$$

which holds for differentiable f and a domain of integration which does not depend on β, we must use the rule:

$$\frac{\partial}{\partial\alpha}\int_{\Omega(\alpha)} f(x,y,\alpha)\;dxdy = \int_{\Omega(\alpha)}\frac{\partial f(x,y,\alpha)}{\partial\alpha}\;dxdy \tag{7.21}$$

$$+ \int_{\Gamma(\alpha)} f(x(s),y(s),\alpha)\;\frac{d\hat{n}(s)}{d\alpha}\;ds. \tag{7.22}$$

Here, the expression $\frac{d\hat{n}(s)}{d\alpha}$ refers to changes in the outward unit normal vector along the boundary $\Gamma(\alpha)$. This formula tells us that a change in β causes changes to the original integral in two ways: the integrand f changes in the original integration region, and the region of integration itself Ω changes. We suggest this situation in Figure 7.2.

Note that this extra term comes about because of the fact that the finite element method uses an integral. If our discrete state equations had been derived from the finite difference method, no such extra term would arise.

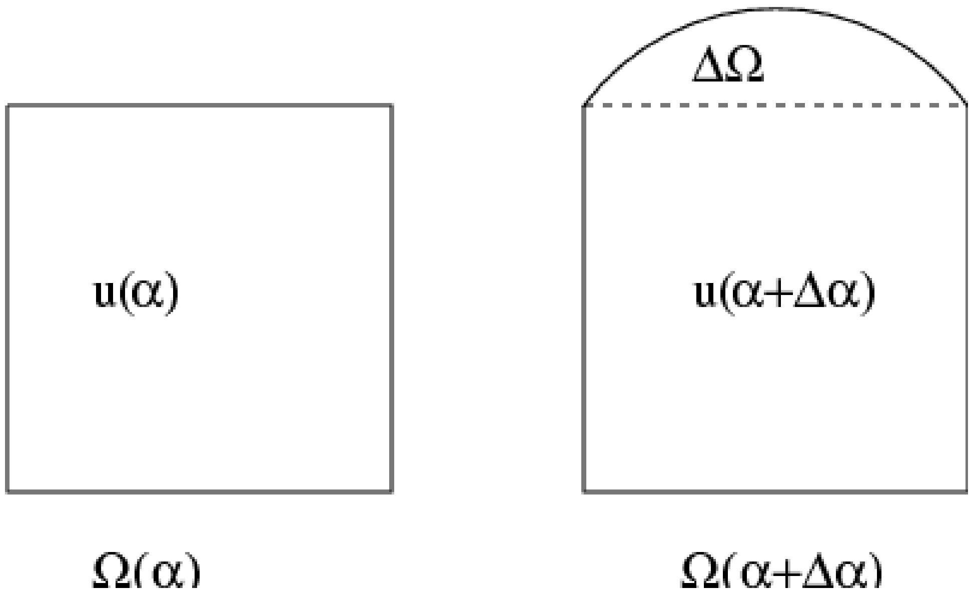

Figure 7.2: How an integral changes when the integrand and region both vary.
We compare changes in u over the original region.
We must also account for changes where the region expands.

With this formula as our guide, we may now proceed to derive the discrete α-sensitivity equations for the discretized finite element state equations. In this set of equations, we have suppressed the indexing of the basis functions w_i and q_i, since we are going to need to take partial derivatives of these functions:

$$\begin{aligned}
\int_{\Omega(\alpha)} (\frac{\partial u^h_\alpha}{\partial x}\frac{\partial w}{\partial x} + \frac{\partial u^h_\alpha}{\partial y}\frac{\partial w}{\partial y} + Re(u^h_\alpha\frac{\partial u^h}{\partial x} + u^h\frac{\partial u^h_\alpha}{\partial x} + v^h_\alpha\frac{\partial u^h}{\partial y} + v^h\frac{\partial u^h_\alpha}{\partial y} + \frac{\partial p^h_\alpha}{\partial x})w)\ d\Omega \ &= \\
-\int_{\Omega(\alpha)} (\frac{\partial u^h}{\partial x}\frac{\partial w_\alpha}{\partial x} + \frac{\partial u^h}{\partial y}\frac{\partial w_\alpha}{\partial y} + Re(u^h\frac{\partial u^h}{\partial x} + u^h\frac{\partial u^h}{\partial x} + v^h\frac{\partial u^h}{\partial y} + v^h\frac{\partial u^h}{\partial y} + \frac{\partial p^h}{\partial x})w_\alpha)\ d\Omega & \\
-\int_{\Gamma(\alpha)} (\frac{\partial u^h}{\partial x}\frac{\partial w}{\partial x} + \frac{\partial u^h}{\partial y}\frac{\partial w}{\partial y} + Re(u^h\frac{\partial u^h}{\partial x} + v^h\frac{\partial u^h}{\partial y} + \frac{\partial p^h}{\partial x})w)\ \frac{d\hat{n}}{d\alpha}\ d\Gamma &
\end{aligned} \tag{7.23}$$

$$\begin{aligned}
&\int_{\Omega(\alpha)} (\frac{\partial v^h_\alpha}{\partial x}\frac{\partial w}{\partial x} + \frac{\partial v^h_\alpha}{\partial y}\frac{\partial w}{\partial y} + Re(u^h_\alpha\frac{\partial v^h}{\partial x} + u^h\frac{\partial v^h_\alpha}{\partial x} + v^h_\alpha\frac{\partial v^h}{\partial y} + v^h\frac{\partial v^h_\alpha}{\partial y} + \frac{\partial p^h_\alpha}{\partial y})w)\ d\Omega \\
&= -\int_{\Omega(\alpha)} (\frac{\partial v^h}{\partial x}\frac{\partial w_\alpha}{\partial x} + \frac{\partial v^h}{\partial y}\frac{\partial w_\alpha}{\partial y} + Re(u^h\frac{\partial v^h}{\partial x} + u^h\frac{\partial v^h}{\partial x} + v^h\frac{\partial v^h}{\partial y} + v^h\frac{\partial v^h}{\partial y} + \frac{\partial p^h}{\partial y})w_\alpha)\ d\Omega \\
&\qquad -\int_{\Gamma(\alpha)} (\frac{\partial v^h}{\partial x}\frac{\partial w}{\partial x} + \frac{\partial v^h}{\partial y}\frac{\partial w}{\partial y} + Re(u^h\frac{\partial v^h}{\partial x} + v^h\frac{\partial v^h}{\partial y} + \frac{\partial p^h}{\partial y})w)\frac{d\hat{n}}{d\alpha}\ d\Gamma
\end{aligned}$$

(7.24)

$$\int_{\Omega(\alpha)} (\frac{\partial u^h_\alpha}{\partial x} + \frac{\partial v^h_\alpha}{\partial y}) q \, d\Omega = - \int_{\Omega(\alpha)} (\frac{\partial u^h}{\partial x} + \frac{\partial v^h}{\partial y}) q_\alpha \, d\Omega - \int_{\Gamma(\alpha)} (\frac{\partial u^h}{\partial x} + \frac{\partial v^h}{\partial y}) q \frac{d\hat{n}}{d\alpha} \, d\Gamma \tag{7.25}$$

The complicated form of these equations may be something of a surprise. Notice, though, that the left hand side, involving the unknowns, has the same form as the discrete sensitivity systems for Re and λ. The complications show up only on the right hand side. Aside from the new boundary integral over Γ, something else has entered the equations: partial derivatives of basis functions w and q with respect to the parameter. This is a peculiarly unwelcome development since it will require a careful analysis of the dependence of the basis functions upon the node locations, and their dependence, in turn, on the value of α.

The boundary conditions are derived in the usual way, that is, by differentiating them, except that the condition on the bump must be deduced as described earlier for the continuous sensitivity equation. The result is:

$$\begin{aligned}
u^h_\alpha(x,y) &= 0 \text{ at inflow and wall velocity nodes;} \\
v^h_\alpha(x,y) &= 0 \text{ at inflow, outflow and wall velocity nodes;} \\
u^h_\alpha(x,y) &= -\frac{\partial u^h}{\partial y} \frac{\partial Bump(x,\alpha)}{\partial \alpha} \text{ at bump velocity nodes;} \\
v^h_\alpha(x,y) &= -\frac{\partial v^h}{\partial y} \frac{\partial Bump(x,\alpha)}{\partial \alpha} \text{ at bump velocity nodes;} \\
p_\alpha(xmax, ymax) &= 0 \text{ at the upper right pressure node.}
\end{aligned}$$

For comparison, we will now discretize the continuous sensitivity equation, that is, reverse the order of the operations of discretization and differentiation. We will see that the resulting equations are significantly different.

7.4 The Discretized α-Sensitivity Equations

To derive the discretized sensitivity equations, we must refer to the continuous sensitivity equations, (7.2), (7.3), and (7.4), and apply the finite element discretization:

$$\begin{aligned}
&\int_{\Omega(\alpha)} (\frac{\partial (u_\alpha)^h}{\partial x}\frac{\partial w_i}{\partial x} + \frac{\partial (u_\alpha)^h}{\partial y}\frac{\partial w_i}{\partial y} \\
&+Re((u_\alpha)^h \frac{\partial u^h}{\partial x} + u^h \frac{\partial (u_\alpha)^h}{\partial x} + (v_\alpha)^h \frac{\partial u^h}{\partial y} + v^h \frac{\partial (u_\alpha)^h}{\partial y} + \frac{\partial (p_\alpha)^h}{\partial x}) w_i)\; d\Omega \;=\; 0
\end{aligned} \tag{7.26}$$

$$\begin{aligned}
&\int_{\Omega(\alpha)} (\frac{\partial (v_\alpha)^h}{\partial x}\frac{\partial w_i}{\partial x} + \frac{\partial (v_\alpha)^h}{\partial y}\frac{\partial w_i}{\partial y} \\
&+Re((u_\alpha)^h \frac{\partial v^h}{\partial x} + u^h \frac{\partial (v_\alpha)^h}{\partial x} + (v_\alpha)^h \frac{\partial v^h}{\partial y} + v^h \frac{\partial (v_\alpha)^h}{\partial y} + \frac{\partial (p_\alpha)^h}{\partial y}) w_i)\; d\Omega \;=\; 0
\end{aligned} \tag{7.27}$$

$$\int_{\Omega(\alpha)} (\frac{\partial (u_\alpha)^h}{\partial x} + \frac{\partial (v_\alpha)^h}{\partial y}) q_i \; d\Omega \;=\; 0 \tag{7.28}$$

The boundary conditions are straightforwardly derived, except that the conditions on the bump are in terms of spatial derivatives of the exact solution u, which we do not know. Therefore, we must approximate those conditions using spatial derivatives of the discretized solution u^h:

$$\begin{aligned}
(u_\alpha)^h(x,y) &= 0 \text{ at inflow and wall velocity nodes;} \\
(v_\alpha)^h(x,y) &= 0 \text{ at inflow, outflow and wall velocity nodes;} \\
(u_\alpha)^h(x,y) &= -\frac{\partial u}{\partial y}\frac{\partial Bump(x,\alpha)}{\partial \alpha} \\
&\approx -\frac{\partial u^h}{\partial y}\frac{\partial Bump(x,\alpha)}{\partial \alpha}
\end{aligned}$$

at bump velocity nodes;

$$(v_\alpha)^h(x,y) = -\frac{\partial v}{\partial y}\frac{\partial Bump(x,\alpha)}{\partial \alpha}$$

$$
\begin{aligned}
& \approx & -\frac{\partial v^h}{\partial y}\frac{\partial Bump(x,\alpha)}{\partial \alpha} \text{ at bump velocity nodes;} \\
(p_\alpha)^h(xmax, ymax) & = & 0 \text{ at the upper right pressure node.}
\end{aligned}
$$

This discretized sensitivity system contrasts strongly with the discrete sensitivity system. In particular, the right hand sides are zero, whereas the right hand sides of the discrete sensitivity system exhibited many terms that will require extensive calculations.

Nonetheless, note that both systems have the same form on the left hand side. Thus, we might say that two systems agree on the operator, but differ on the right hand side.

Finally we note that, for the discretized sensitivity equations, the boundary conditions along the bump are a primary source of error. To apply the boundary condition exactly, we need to know the partial derivatives of the *true solution* u and v. In fact, we only have knowledge of the discretized solution variables u^h and v^h, and our approximation is made worse by the fact that we are working at points on the boundary, and because we are approximating a spatial derivative rather than a state variable. Roughly speaking, we have seen that the true velocities u are approximated by the Taylor Hood finite element method with an error which we estimate to be of order $O(h^2)$. We said that this meant that a spatial derivative like $\frac{\partial u}{\partial y}$ would be approximated with an error of $O(h)$. This means that there is an error of order $O(h)$ in our specification of the boundary condition for the discretized shape sensitivities u_α^h.

These problems do not occur for the discrete sensitivity equations, because there the boundary conditions are given in terms of the discrete solution, which is known exactly.

7.5 Exact Example: Shape Sensitivity of Poiseuille Flow

Let us try to get our bearings by working with the simple case of Poiseuille flow.

Since we can only produce exact solutions of the Poiseuille flow when the channel is rectangular, we don't want to insert a curved bump into the channel. Instead, our parameter α will move the upper wall up and down, as a whole. Thus, the value of α will be the coordinate of the upper wall.

We assume an inflow profile of the form:

$$Inflow(y, \alpha, \lambda) = \lambda y(\alpha - y). \tag{7.29}$$

For any $0 < \alpha$, we can write out the exact continuous state solution at every point in the domain:

$$u(x, y, \alpha) = \lambda y(\alpha - y) \tag{7.30}$$

$$v(x, y, \alpha) = 0 \tag{7.31}$$

$$p(x, y, \alpha) = 2\lambda(xmax - x)/Re \tag{7.32}$$

and we can write the sensitivities with respect to the implicit shape parameter α:

$$u_\alpha(x, y, \alpha) = -\lambda y \tag{7.33}$$

$$v_\alpha(x, y, \alpha) = 0 \tag{7.34}$$

$$p_\alpha(x, y, \alpha) = 0 \tag{7.35}$$

Note that, for this special problem, the $Inflow$ function also has a dependence on α, and that we will make use of the fact that for this simple problem,

$$\frac{\partial Bump}{\partial \alpha} = 1. \tag{7.36}$$

The continuous α-sensitivity equations for this problem are:

$$-(\frac{\partial^2 u_\alpha}{\partial x^2} + \frac{\partial^2 u_\alpha}{\partial y^2}) + Re(u_\alpha \frac{\partial u}{\partial x} + u\frac{\partial u_\alpha}{\partial x} + v_\alpha \frac{\partial u}{\partial y} + v\frac{\partial u_\alpha}{\partial y} + \frac{\partial p_\alpha}{\partial x}) = 0 \tag{7.37}$$

$$-(\frac{\partial^2 v_\alpha}{\partial x^2} + \frac{\partial^2 v_\alpha}{\partial y^2}) + Re(u_\alpha \frac{\partial v}{\partial x} + u\frac{\partial v_\alpha}{\partial x} + v_\alpha \frac{\partial v}{\partial y} + v\frac{\partial v_\alpha}{\partial y} + \frac{\partial p_\alpha}{\partial y}) = 0 \tag{7.38}$$

$$\frac{\partial u_\alpha}{\partial x} + \frac{\partial v_\alpha}{\partial y} = 0 \tag{7.39}$$

with the boundary conditions:

$$
\begin{aligned}
u_\alpha(0,y) &= \frac{\partial Inflow(y,\alpha,\lambda)}{\partial \alpha} \\
&= \lambda y; \\
v_\alpha(0,y) &= 0; \\
u_\alpha(x,y) &= -\frac{\partial u}{\partial y} \\
&= -\lambda y \text{ on the upper wall;} \\
v_\alpha(x,y) &= -\frac{\partial v}{\partial y} = 0 \text{ on the upper wall;} \\
u_\alpha(x,y) &= v_\alpha(x,y) = 0 \text{ on the lower wall;} \\
\frac{\partial u_\alpha}{\partial x}(xmax,y) &= 0 \text{ on the outflow;} \\
v_\alpha(xmax,y) &= 0 \text{ on the outflow;} \\
p_\alpha(xmax,ymax) &= 0.
\end{aligned}
$$

We can easily verify that this system is satisfied by the sensitivities that we would get by differentiating the exact solution:

$$u_\alpha(x,y,\alpha) = \lambda y \tag{7.40}$$

$$v_\alpha(x,y,\alpha) = 0 \tag{7.41}$$

$$p_\alpha(x,y,\alpha) = 0 \tag{7.42}$$

Thus, we have shown that we can derive the continuous sensitivity equations for the Poiseuille flow with a moving wall, and compute values for the sensitivities which are equal to the known, exact values.

We should note that the shape parameter required us not only to add a new boundary condition along the moving shape, but also had an effect on the form of the inflow boundary condition. This happened because the shape parameter affected the region where the inflow

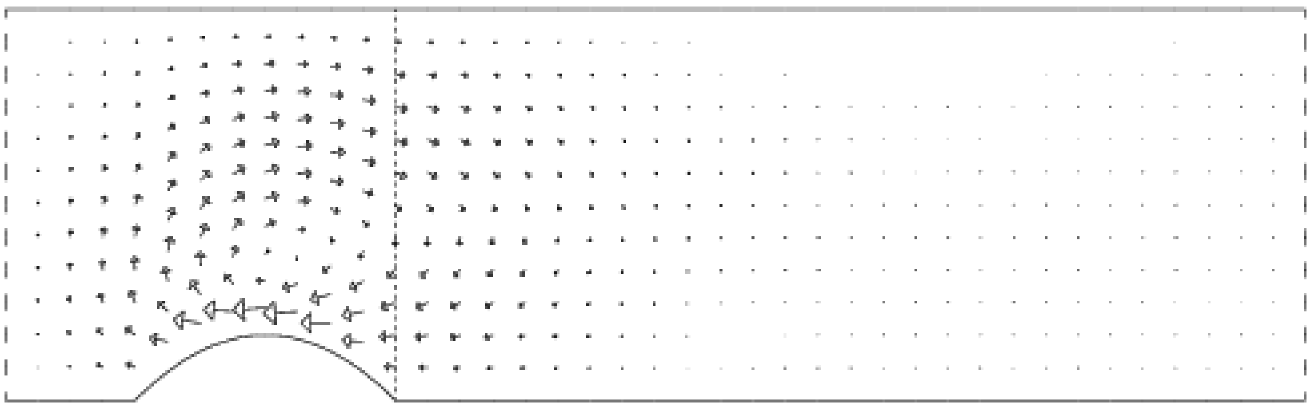

Figure 7.3: Discretized velocity α-sensitivity.

boundary condition was specified. Such secondary effects of a shape parameter may be hard to foresee or calculate in general.

7.6 Computational Example: Flow Past a Bump

If we return to the three-parameter problem already discussed in section 6.6, we may now compute the discretized flow sensitivities with respect to the parameter α that affects the flow variables implicitly by determining the shape of the bump that intrudes into the flow region.

Now we can, if we wish, plot the discretized velocity sensitivity field as though it were a physical quantity, as in Figure 7.3. In this case, a plot vector drawn at a particular point represents the relative change that would apply to the current velocity at that point, with a unit change in α. The discretized sensitivity velocity vectors don't behave like a physical flow: they satisfy the Oseen equations rather than the Navier Stokes equations. However, the same continuity equation appears in both sets of equations, and we can see directly from

Table 7.1: Finite difference check of discretized α-sensitivities.

Variable	$\|(u_\alpha)^h\|_\infty$	$\|\Delta(u^h)/\Delta(\alpha)\|_\infty$	$\|\text{Difference}\|_\infty$
U	0.6541	0.8046	1.5E-01
V	0.2709	0.2702	1.2E-01
P	0.4709	0.4378	2.3E-01

the plot that the continuity equation seems to be satisfied: the vector quantity represented by the plotted arrows seems to satisfy the rule that "what comes in goes out".

We can glean other useful information from this figure. The plot shows that the influence of the shape parameter is restricted to the area near the bump, generating a sort of "whorl". Further downstream, the influence is negligible. This means that, unlike the inflow parameter, measurements of the effect of the shape parameter should be made near to the bump. We will see the importance of this fact later, when we try to use a profile line that is too far downstream from the bump.

7.7 Comparison of Finite Differences and Discretized Sensitivities

We return to the three-parameter problem discussed in Section 6.7. We would like to make a similar comparison, between the discretized sensitivities $(u_\alpha)^h$ and estimates of the influence of α made using finite differences. As in the previous table, we will simply compare the vector of finite element coefficients for $(u_\alpha)^h$ with the vector of differences in the finite element coefficients for u at α and at $\alpha + \Delta\alpha$, divided by $\Delta\alpha$. Table 7.1 reports the maximum norm of these two vectors, and of their difference. As in the previous Tables 6.1 and 6.2, our mesh parameter is $h = 0.25$. The perturbation $\Delta(\alpha)$ was computed using a formula similar to Equation (6.46).

The discrepancy here is startling, particularly when compared to the near perfect agreement obtained for the λ and Re sensitivities. There are actually several factors to investigate. First, we should already be aware that we are using discretized sensitivities, rather than discrete sensitivities. Secondly, as we will discover, our finite difference results are comparing the same coefficient at different values of α. If α were not a shape variable, then the two values of the coefficient would both be associated with the same fixed spatial value. But because we regrid the region when we compute the solution at $\alpha + \Delta\alpha$, our two coefficient values are actually associated with different positions, which is not a proper approximation to the *partial* derivative of $u^h(x, y, \alpha)$, which holds x and y fixed.

The proper computation of finite difference estimates of the sensitivities, and their comparison to discretized sensitivities, is a complicated matter that we will analyze carefully in the next chapter. At the end of that chapter, we will again make a comparison chart like Table 7.1, but we will have better agreement, and be able to explain what is going on.

Chapter 8

FINITE DIFFERENCE ESTIMATES OF SENSITIVITIES

8.1 Introduction

A common method of approximating derivatives is to use finite differences. Since the sensitivities are derivatives, or approximations of them, we would like to compute finite difference estimates of the quantities whose sensitivites we are studying. Such an independent computation will be useful as a check of both the accuracy and efficiency of the discretized sensitivity approach.

We compare the definitions, computations, accuracy, and usefulness of discretized sensitivities and finite difference sensitivities. We show that if shape parameters are involved, it is more difficult to make comparisons, particularly if we are interested in comparing individual finite element coefficients, that is, values associated with nodes. This is mainly because the nodes move, while a sensitivity must refer to changes at a fixed location.

We consider an adjustment to finite difference calculations involving finite element coefficients. This adjustment allows a comparison with the discretized sensitivities. We show data that suggests that, for our finite element formulation, the approximation error, as measured by the difference between the discretized sensitivities and the adjusted finite difference quantities, decreases linearly with h.

This is empirical evidence that the discretized sensitivities, which are cheaper to compute than discrete sensitivities, have good enough approximation abilities for most uses.

8.2 Finite Coefficient Differences and Finite Physical Differences

In earlier discussions of sensitivities with respect to a parameter, we have mentioned the idea of comparing a discretized sensitivity and a finite difference estimate. We did not stop to explain this concept; it is a common technique, and, for explicit parameters, does not require any special thought or consideration.

However, for a geometric parameter, particularly in the discrete case, we will see that the calculation of a finite difference sensitivity, and comparison to a discretized sensitivity requires some extra care. Let us suppose that, for some geometric parameter value α, we have a solution to the discrete problem, comprising a set of finite element coefficients. To be more particular, let us suppose that one of those items is u_i, a velocity coefficient associated with node i, which is itself located at node (x_i, y_i).

If we change the value of α, the quantities that change may include not just u_i, but also the location of the associated node. At this point there are two paths we could follow. If we simply compute the finite difference quotient

$$\frac{u_i(\alpha + \delta\alpha) - u_i(\alpha)}{\delta\alpha} \tag{8.1}$$

then we are approximating the behavior of the finite element coefficient itself, which we can term a *finite coefficient difference*.

To approximate the behavior of the the physical quantity u at (x_i, y_i), we would need to determine where this point lies in the finite element mesh associated with $\alpha + \delta\alpha$, evaluate the finite element approximation, and then compute the quotient:

$$\frac{u^h(x_i, y_i, \alpha + \delta\alpha) - u^h(x_i, y_i, \alpha)}{\delta\alpha}. \tag{8.2}$$

This is a proper approximation of the physical quantity, and hence we might term this a *finite physical difference.*

It may not be clear, but it is much easier to compute the finite coefficient differences than the finite physical differences. As a geometric parameter varies, the nodes and elements move, complicating the evaluation of the perturbed solution. But the structure of the finite element coefficient vector itself will generally not change, and it is a trivial matter to compute differences of comparable vector elements.

To get an idea of the distinction between these quantities, we will look at plots of the discretized sensitivities (which approximate the same quantities as the finite physical differences) versus the finite coefficient differences. Figure 8.1 displays the discretized α-sensitivities for velocity, while Figure 8.2 shows a corresponding finite coefficient difference field, computed by comparing coefficient values.

Significant information can be found by comparing the two plots. In particular, we note that the discretized sensitivity field is nonzero on the bump, whereas the corresponding finite difference sensitivities vanish; the discretized sensitivities seem to satisfy the continuity equation (the discretized sensitivity equation forces this!) while this is not true for the finite difference sensitivities; the discretized sensitivities and finite difference sensitivities seem to agree in the portions of the region before and after the bump, where the geometry is never distorted.

The plot should make it clear that discretized sensitivities and finite coefficient differences can

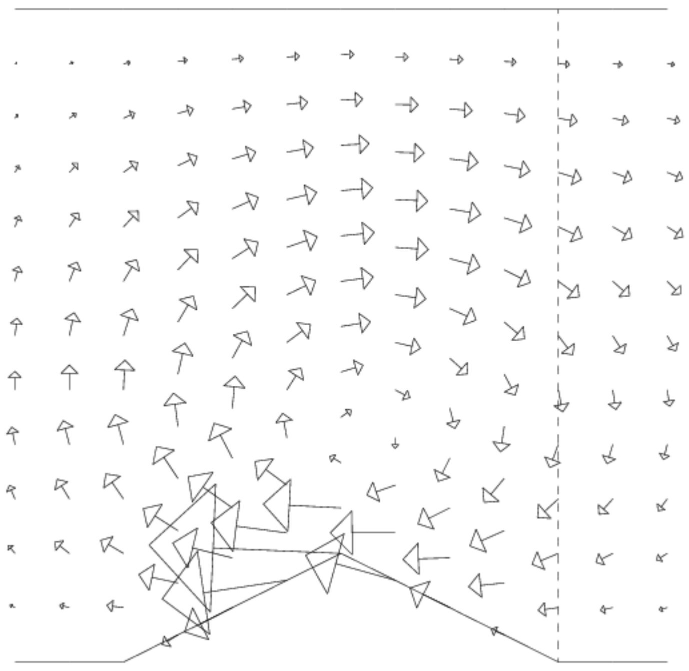

Figure 8.1: Discretized α-sensitivity nodal values.
Notice, in particular, the nonzero boundary condition on the bump.
The sensitivities estimate change at a fixed location.

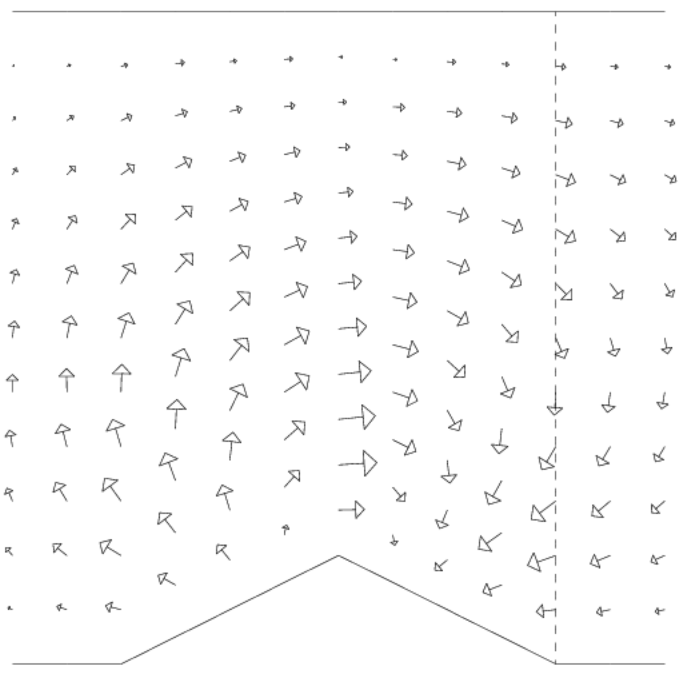

Figure 8.2: The corresponding α-finite difference nodal values.
The boundary values are exactly zero, and the "vortex" is quite different.
These values apply at a fixed node, but a moving location.

be quite different quantities. We propose to explain what this difference is, and how it can be quantified. We are interested in how well our discretized sensitivities approximate the true continuous sensitivities and the discrete sensitivities. Understanding the finite coefficient differences will help us in this task.

8.3 Differencing with Respect to Geometric Parameters

Throughout this chapter, we will assume that we are varying a single shape parameter which we shall denote by α. We will occasionally want to write a formula that references state variable values such as $u(x, y)$. When we are varying a geometric parameter, there may be times when the point (x, y) does not lie within the current flow region $\Omega(\alpha)$, so that we don't have a well-defined value for u. In such cases, let us agree that u will simply be set to 0. While better choices exist to extend the solution to the neighboring area, right now we are only interested in assuring ourselves that an expression like $u(x, y)$ will always have a defined value.

In order to distinguish the information produced by a finite coefficient difference sensitivity from that in a discretized sensitivity, we must consider how a finite difference quotient is computed. Again, we assume that we have a particular value of α of interest to us, with a corresponding solution (u, v, p), and as usual, we will focus our attention on u. Typically, once the current solution u is calculated at α, we slightly perturb the value of the parameter to $\alpha + \Delta\alpha$ and compute a new solution, which we may write $u(\alpha + \Delta\alpha)$. We then form a difference quotient whose numerator is the change in the solution at a fixed location (x, y), and whose denominator is the change in the parameter. Since the (total) derivative of a quantity depending on α is defined as the limit of such quotients as $\Delta\alpha \to 0$, we can expect, for reasonably smooth problems and small $\Delta\alpha$, that we can compute a good approximation to this quantity.

If we focus our attention on a point (x, y) which lies in $\Omega(\alpha)$, and consider how the value of $u(x, y)$ will vary with α, then we can approximate the desired partial derivative $\frac{\partial u}{\partial \alpha}(x, y, \alpha)$ by the finite difference quotient:

$$\frac{\Delta u(x, y, \alpha)}{\Delta\alpha} \equiv \frac{u(x, y, \alpha \pm \Delta\alpha) - u(x, y, \alpha)}{\pm\Delta\alpha} \tag{8.3}$$

Perhaps we should explain the cumbersome form of the quotient on the right hand side. In this formula, we assume that the perturbation $\Delta\alpha$ is small enough, and that we have chosen the proper sign for it, so that the point (x, y) is not only in $\Omega(\alpha)$ but also in $\Omega(\alpha + \Delta\alpha)$. Simple control of the size of the perturbation $\Delta\alpha$ is enough to guarantee that a point (x, y) in the interior of $\Omega(\alpha)$ will also lie in the interior of $\Omega(\alpha + \Delta\alpha)$. But for points that actually lie on the bump, we need to control the sign of the perturbation as well, and even then, we can't guarantee that the point will remain in the perturbed region unless we can show that:

$$\frac{\partial Bump}{\partial \alpha}(x, \alpha) \neq 0. \tag{8.4}$$

For a bump defined by a piecewise linear function, for instance, we can guarantee that Condition (8.4) is true everywhere except at the fixed endpoints, where we don't care.

So now if u is the solution of a continuous problem, Equation (8.3) may be used to define a finite difference quotient $\frac{\Delta u}{\Delta \alpha}(x, y, \alpha)$ which has values throughout the flow region, and which estimates the influence of α. The sensitivity of the solution of the continuous problem at (x, y) is the partial derivative $\frac{\partial u(x, y, \alpha)}{\partial \alpha}$, but this derivative is precisely the limit of the finite difference quotients defined by Equation (8.3), so we may assert immediately that, for the continuous problem,

$$\lim_{\Delta\alpha \to 0} \frac{\Delta u}{\Delta \alpha}(x, y, \alpha) = u_\alpha(x, y, \alpha), \tag{8.5}$$

and thus, finite difference quotients can be used as a check on the accuracy of the sensitivity calculation.

Now, by contrast, let us suppose that we are interested in the value of u at a *moving* point. This might seem a needless complication, until one realizes that points on the bump surface move, that we have nodes on the bump, and in our formulation, all the interior nodes above the bump will also move if the bump moves. The interior nodes determine the shape of the elements, the form of the basis functions, and the physical "anchor" for the coefficient values, so a lot changes when a node moves.

Both on the bump, and at the interior nodes, we expect to find nonzero sensitivities. We will keep our problem simple by assuming that only the y coordinate of such a point varies with α, so that we may write its coordinates as $(x, y(\alpha))$. Further, we assume that this point remains in $\Omega(\alpha)$ for all values of α of interest to us. At such a moving point, we now must consider u to be effectively a function of only x and α. To study the influence of the parameter α, we would need to find the total derivative of u with respect to α, which we write $\dfrac{Du}{D\alpha}(x, y(\alpha), \alpha)$, and we may approximate this total derivative by the finite difference quotient:

$$\frac{\Delta u(x, y(\alpha), \alpha)}{\Delta\alpha} = \frac{u(x, y(\alpha + \Delta\alpha), \alpha + \Delta\alpha) - u(x, y(\alpha), \alpha)}{\Delta\alpha}. \tag{8.6}$$

Moreover, we note that for this situation, we have a simple relationship between the total derivative and the sensitivity:

$$\frac{Du}{D\alpha}(x, y(\alpha), \alpha) = \frac{\partial u}{\partial y}(x, y(\alpha), \alpha)\frac{dy(\alpha)}{d\alpha} + \frac{\partial u}{\partial\alpha}(x, y(\alpha), \alpha), \tag{8.7}$$

which breaks down our total derivative into a change due to spatial movement, and a change due to variations in α.

From what we have discussed so far, we can now explain a few things about Figure 8.1 and Figure 8.2. First of all, in the regions before and after the bump, none of the nodes move with changes in α. Therefore, in this region the discretized sensitivities and the finite coefficient differences should agree closely.

We can also explain the "boundary value discrepancy", that is, why on the surface of the bump the discretized sensitivities are nonzero while the finite coefficient differences are exactly zero. The discretized sensitivity approximates the true sensitivity, which is computed as the limit of finite difference quotients that compare the value of u to values of u for nearby values of α, but at the same location. Clearly, if α decreases, the velocity at the point (which is now zero) will increase, that is, fluid will pass through the point, moving to the right. Therefore, the sensitivity, which is the limit of ratios of this positive change in u to this negative change in α is represented as a *negative* horizontal velocity increment.

By contrast, the finite coefficient difference compares the current value of u (which is zero) to values of u at nearby values of α, at locations that move (up or down) with the bump. But a point on the bump always has zero horizontal velocity u, so the finite coefficient difference associated with a node on the bump will always be zero.

Thus, we arrive at quite different results, based on whether we allow the point under consideration to move with α or stay fixed.

8.4 Finite Differences for a Discretized Problem

Now let us suppose that, corresponding to a parameter value of α, we have the solution of a discrete problem, which may be represented a set of coefficient data $u_i^h(\alpha)$, which in turn determine a continuously differentiable function of x, y, and α:

$$u^h(x,y,\alpha) \equiv \sum_i u_i^h(\alpha)\ w_i(x,y,\alpha). \tag{8.8}$$

where, as usual, if the geometric point of evaluation happens to be the i-th node, (x_i, y_i), then

$$u^h(x, y_i(\alpha), \alpha) = u_i^h(\alpha). \tag{8.9}$$

As in the continuous case, we may define the following finite difference approximation to the partial derivative $\dfrac{\partial u^h(x,y,\alpha)}{\partial \alpha}$:

$$\frac{\Delta u^h(x,y,\alpha)}{\Delta \alpha} \equiv \frac{u^h(x,y,\alpha \pm \Delta\alpha) - u^h(x,y,\alpha)}{\pm\Delta\alpha}. \tag{8.10}$$

where we choose the magnitude of $\Delta\alpha$ and, if necessary, the sign, so that the point (x, y) is in $\Omega(\alpha + \Delta\alpha)$ as well as in $\Omega(\alpha)$.

From this definition, and assuming u^h is twice continuously differentiable in α, we may use a Taylor expansion to see that

$$\frac{\Delta u^h(x,y,\alpha)}{\Delta \alpha} = \frac{\partial u^h(x,y,\alpha)}{\partial \alpha} + O(\Delta\alpha), \tag{8.11}$$

and hence, for small $\Delta\alpha$, the finite difference quotient is a good approximation to the sensitivity.

Now, if we again assume we are interested in the changes in u^h at a point whose y-coordinate changes with α, and if we assume that the point is always in the flow region, then we consider the difference quotient:

$$\frac{\Delta u^h(x, y(\alpha), \alpha)}{\Delta\alpha} = \frac{u^h(x, y(\alpha + \Delta\alpha), \alpha + \Delta\alpha) - u^h(x, y(\alpha), \alpha)}{\Delta\alpha}, \tag{8.12}$$

and we can use a Taylor expansion to show that

$$\begin{aligned} \frac{\Delta u^h(x, y(\alpha), \alpha)}{\Delta\alpha} &= \frac{Du^h(x, y(\alpha), \alpha)}{D\alpha} + O(\Delta\alpha) & (8.13)\\ &= \frac{\partial u^h}{\partial y}\frac{dy}{d\alpha} + \frac{\partial u^h}{\partial\alpha} + O(\Delta\alpha). & (8.14) \end{aligned}$$

In particular, let us take the moving point to be velocity node i, so that the y-coordinate of the node may be written $y_i(\alpha)$. In this case, we may write

$$\begin{aligned} \frac{\Delta u^h(x, y_i(\alpha), \alpha)}{\Delta\alpha} &= \frac{u^h(x, y_i(\alpha + \Delta\alpha), \alpha + \Delta\alpha) - u^h(x, y_i(\alpha), \alpha)}{\Delta\alpha} & (8.15)\\ &= \frac{u_i^h(\alpha + \Delta\alpha) - u_i^h(\alpha)}{\Delta\alpha}, & (8.16) \end{aligned}$$

where we have written $u_i^h(\alpha)$ to indicate the dependence of the i-th horizontal velocity finite element coefficient on the parameter α. In other words, the total derivative of u^h along the path of the moving node is simply the difference quotient of the associated coefficient $u_i^h(\alpha)$.

Combining Equations (8.14) and (8.16), we see that:

$$\frac{u_i^h(\alpha + \Delta\alpha) - u_i^h(\alpha)}{\Delta\alpha} = \frac{\partial u^h}{\partial y}(x, y_i(\alpha), \alpha)\frac{dy_i(\alpha)}{d\alpha} + \frac{\partial u^h}{\partial\alpha}(x, y_i(\alpha), \alpha) + O(\Delta\alpha), \tag{8.17}$$

which means that we may estimate the true sensitivity at node i by

$$\begin{aligned} u_\alpha^h(x, y_i(\alpha), \alpha) &\equiv \frac{\partial u^h(x, y_i(\alpha), \alpha)}{\partial\alpha} & (8.18)\\ &\approx \frac{u_i^h(\alpha + \Delta\alpha) - u_i^h(\alpha)}{\Delta\alpha} - \frac{\partial u^h}{\partial y}(x, y_i(\alpha), \alpha)\frac{dy_i(\alpha)}{d\alpha}. & (8.19) \end{aligned}$$

Now, if we call the quantity on the right hand side of Equation (8.19) the "adjusted finite coefficient difference", we can now see how to relate three quantities of interest to us: the discretized sensitivities, the finite coefficient differences, and the discrete sensitivities. We will make our comparison by considering individual finite element coefficients. For a particular value of α, consider a flow solution (u^h, v^h, p^h), and some finite element node i set at $(x, y_i(\alpha))$. The discrete sensitivity of the solution is $u^h_\alpha(x, y_i(\alpha))$. We can now see that this quantity is approximated to order $O(\Delta\alpha)$ by the adjusted finite coefficient difference. Moreover, the discrete sensitivity at $(x, y_i(\alpha))$ is also approximated, to an order depending in some way on the mesh parameter h (which we expect to be roughly $O(h)$ for our Taylor Hood approach), by the discretized sensitivity $(u_\alpha)^h(x, y_i(\alpha))$, which is simply the discretized sensitivity coefficient at node i. Thus, for suitably small values of $\Delta\alpha$ and h, the discretized sensitivity coefficient and the adjusted finite coefficient difference should be comparable.

Therefore, for a discrete problem, we can compare discretized sensitivities and finite difference sensitivities at a node by comparing the corresponding finite element coefficients, but we must make an adjustment to the finite difference sensitivity coefficients in cases where the associated node actually moves with changes in α.

To demonstrate this fact, we adjust the finite coefficient difference velocity field that was shown Figure 8.2 (and which matched so poorly with our discretized sensitivity field) and display the result in Figure 8.3. There are no arrows displayed along the boundary, because we don't have unknown coefficients associated with those nodes. In the interior, however, the agreement with the discretized velocity sensitivity field of Figure 8.1 is now striking.

Equation (8.18) tells us how to adjust the finite coefficient differences associated with a moving node, so that they approximate the sensitivity of a solution quantity at a fixed point. Conversely, the equation also tells us how to transform the discretized sensitivity coefficient associated with a node so that we can estimate the change in the solution coefficient as the

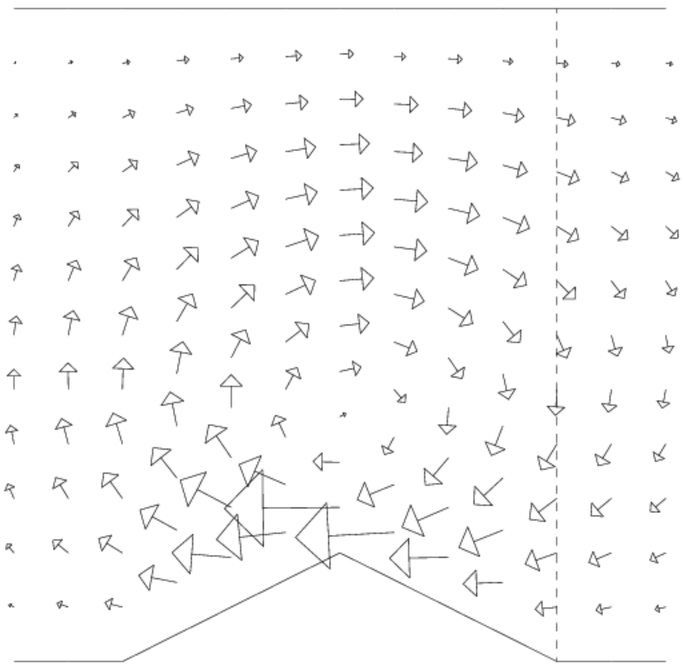

Figure 8.3: Adjusted finite difference α-sensitivity nodal values.

parameter changes and the node moves. The corresponding formula is simply:

$$\frac{u_i^h(\alpha+\Delta\alpha)-u_i^h(\alpha)}{\Delta\alpha} \approx (u_\alpha)^h(x,y_i(\alpha),\alpha)+\frac{\partial u^h}{\partial y}(x,y_i(\alpha),\alpha)\frac{dy_i(\alpha)}{d\alpha}. \tag{8.20}$$

Whereas Equation (8.18) is useful for estimating the closeness of our approximation to the true sensitivities, Equation (8.20) is useful if we wish to use discretized sensitivities to estimate the change that will occur in our finite element coefficients associated with a moving node, when a shape parameter is varied. This is exactly the sort of calculation we must make when trying to produce a suitable starting point for the Picard or Newton iteration.

8.5 Sensitivity Approximation with Decreasing h

We now wish to make a comparison, for a shape parameter, between the discretized sensitivities and the adjusted and unadjusted finite coefficient differences. We already saw a fairly poor agreement in the previous chapter, where we compared discretized sensitivities and unadjusted finite coefficient differences.

Table 8.1 is based on solutions of a problem involving three parameters, (λ, α, Re), and a flow solution computed at $(0.5, 0.5, 10.0)$, on grids with $h = 0.25$, $h = 0.125$, and $h = 0.0625$. For each mesh parameter, we computed the discretized sensitivities, the finite coefficient differences, and the adjusted finite coefficient differences, that is, the results of Equation (8.18). In Table 8.1 we print out the largest of each of these quantities, broken down into the components associated with u, v and p. The last column gives the maximum difference between the finite element coefficients for the discretized sensitivities and the adjusted finite coefficient differences.

From the table, it is clear that we are not getting the superb agreement between discretized sensitivities and finite coefficient differences that we have seen for explicit parameters; even

Table 8.1: Finite coefficient differences versus discretized α-sensitivities. Table entries represent the maximum absolute coefficient value.

h	Finite Coef Difference (FCD)	Adjusted FCD (AFCD)	Discretized Sensitivity (DS)	AFCD - DS
		U		
0.25	0.309	0.802	0.654	0.148
0.125	0.326	1.07	1.02	0.0834
0.0625	0.327	1.33	1.30	0.0327
		V		
0.25	0.242	0.260	0.271	0.108
0.125	0.260	0.273	0.272	0.0584
0.0625	0.258	0.276	0.272	0.0232
		P		
0.25	0.438	0.438	0.471	0.133
0.125	0.400	0.493	0.507	0.0684
0.0625	0.346	0.462	0.476	0.0338

the adjusted finite coefficient differences have a significant disagreement with the discretized sensitivities. However, we expected that fact. The first interesting thing to note is how much better the agreement is between the discretized sensitivities and the adjusted finite coefficient differences. Evidently, the adjustment for moving nodes is quite significant. The more important question is, how does the disagreement behave with decreasing mesh parameter h?

We ask this question because we really want to know how the discretized sensitivities approximate the true sensitivities. That information is not available to us, since we have not tried to compute the true sensitivities. However, we assume that the adjusted finite coefficient differences are a good estimate of the true sensitivities. If the discretized sensitivities approximate the adjusted finite coefficient differences well, then we expect that they are almost as good an approximation of the true sensitivities.

We must keep in mind, though, that the adjusted finite coefficient differences are by no

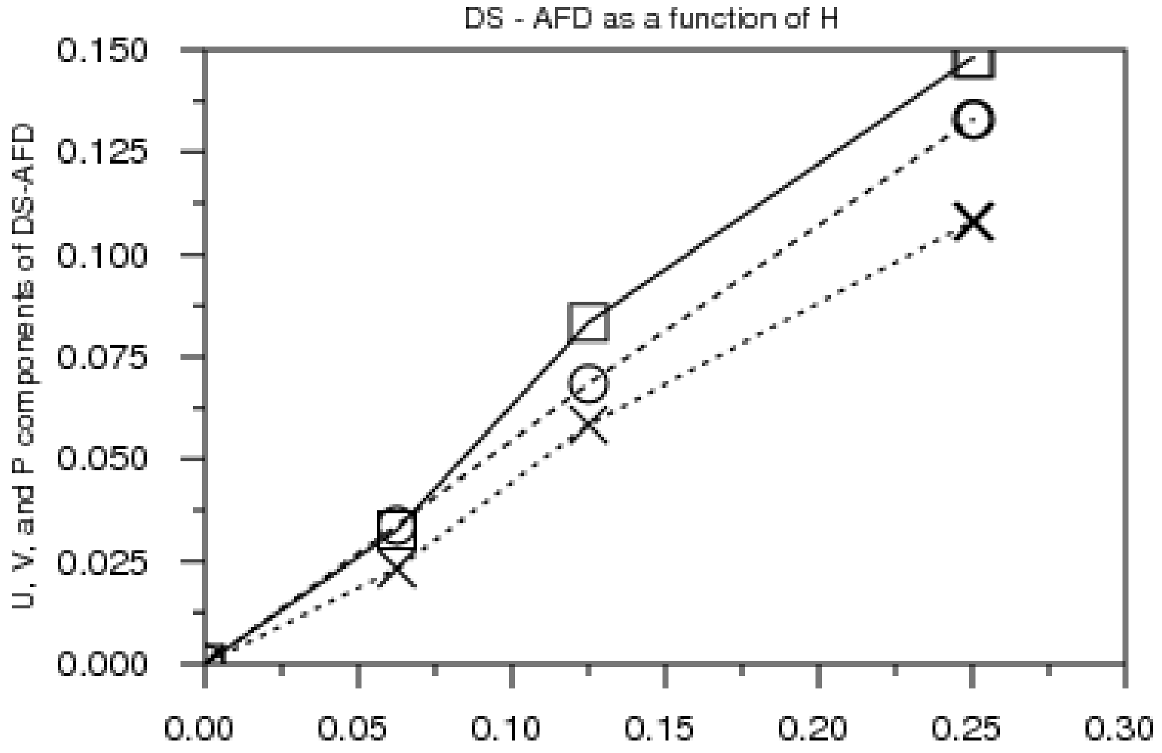

Figure 8.4: Discretized sensitivity/adjusted finite coefficient difference discrepancies. The box marks u data, the cross v, the circle p.

means perfect estimates of the true sensitivities. The errors incurred by the adjusted finite coefficient differences include the usual finite difference error, but also the added inaccuracy due to the use of the Taylor approximation involving $\frac{\partial u^h}{\partial y}$ for the adjustment term.

The discrepancies between the discretized sensitivities and the adjusted finite coefficient differences are plotted in Figure 8.4. There is a clear trend in the data. For each solution component, the discrepancy between the discretized sensitivity and the adjusted finite coefficient differences drops roughly in step with h; as h is halved, so is the discrepancy. This bears out our claim that, even for geometric parameters, the discretized sensitivities converge to the true sensitivities as h goes to zero. The actual order of convergence depends, of course, on the particular discretization applied to the problem, and seems, for our problem, to be behaving linearly in h, as we have already estimated.

The table contains more information, if we look closely. As h decreases, the infinity norms of the horizontal velocity and the pressures increase significantly. This should be surprising. It suggests that the grid is still not fine enough to capture important features of the flow. But this hasn't happened on other problems, so what is different here? The difference is surely the fact that there is a strong boundary condition being applied on a curved boundary. Very important effects are occurring just on the boundary, and we are missing some of these details if the mesh is not fine enough.

Chapter 9

OPTIMIZATION AND THE COST FUNCTIONAL

9.1 Introduction

Chapter 1 introduced the problem of finding parameter values producing a flow solution that best matched a given set of flow data along a profile line. Subsequent chapters have developed the tools needed to set up parameters, define the problem, and determine the corresponding flow pattern.

Now that we can produce flow solutions, we want a way to assign a *score* that evaluates how well that solution satisfies our requirements. The measurement we will use will be defined in a *cost functional.* Once the cost functional is specified, our problem is ready for automatic treament by an *optimization algorithm*, which determines how to go about choosing particular sets of parameters to test by evaluating the cost functional.

We have seen how the flow parameters β determine the particular problem to be solved, and how, at least locally, the resulting flow solution (u, v, p) can be regarded as a function of these parameters. We will see that the cost functional, initially defined in terms of the state variables, can also be regarded as a function of the underlying parameters.

The optimization algorithm will generally require not only the evaluation of the cost functional for a given set of parameters, but also the partial derivatives of the cost functional with respect to those parameters. We will investigate how to make this calculation, given that we don't have the necessary explicit information.

In anticipation of problems that will arise during computation, we discuss some modifications that can be made to the cost functional, involving *penalty functions*, in pursuit of *regularization*. Reasons for modifying the cost functional in this way include the existence of local minima or a functional whose contour levels are very "twisted" or badly scaled.

9.2 Local Minimizers in Computational Optimization

A *local minimizer* of a functional is a point which has a functional value that is less than or equal to the functional value of every other point in some sufficiently small neighborhood. A local minimizer may be contrasted with a *global minimizer*, whose functional value is less than or equal to the functional value of *all* other points. While a global minimizer is obviously more desirable than a local minimizer, most practical optimization algorithms are only able to locate local minimizers.

Some methods for checking whether a point is actually a true local minimizer are not suitable for a practical, finite computation: we might wish to have a perfectly true graph of the functional, in which case a single glance would tell us everything we need to know; we might imagine evaluating the functional at every point in a neighborhood of the candidate; we might, if we had an explicit formula for the functional in terms of the parameters, perform some simple analysis on the gradient.

The optimization code has no such luxuries. Its analysis of the problem is based entirely on sampling the cost functional and its gradient (which may itself be inaccurate!) at a finite number of points. The Hessian matrix is also useful in this regard, and an optimization code may desire values of this matrix from the user, or, as in our case, attempt to approximate the matrix based on functional and gradient information. The following theorems suggest how this information may be used in the search for local minimizers (Dennis and Schnabel [10]):

Theorem 9.1 (The Gradient of a Local Minimizer) *Suppose that $\mathcal{J} : R^n \rightarrow R$ is continuously differentiable in an open neighborhood of β_0. If β_0 is a local minimizer of $\mathcal{J}(\beta)$, the gradient vector $\nabla\mathcal{J}(\beta)$ must vanish at β_0.*

Theorem 9.2 (The Hessian of a Local Minimizer) *Suppose that $\mathcal{J} : R^n \rightarrow R$ is twice continuously differentiable in an open convex neighborhood of β_0. If β_0 is a local minimizer of $\mathcal{J}(\beta)$, and if the Hessian matrix $\nabla^2\mathcal{J}(\beta)$ is Lipschitz continuous at β_0, then the Hessian matrix must be positive semidefinite at β_0.*

If we strengthen the condition on $\nabla^2\mathcal{J}(\beta)$, we can turn these two necessary conditions on a minimizer into a sufficient condition for local minimization:

Theorem 9.3 (A Test for Local Minimization) *Suppose that $\mathcal{J} : R^n \rightarrow R$ is twice continuously differentiable in an open convex neighborhood of β_0, that the gradient vector $\nabla\mathcal{J}(\beta)$ vanishes at β_0, and that the Hessian matrix $\nabla^2\mathcal{J}(\beta)$ is Lipschitz continuous and nonsingular at β_0.*

Then β_0 is a local minimizer of $\mathcal{J}$ if, and only if, $\nabla^2\mathcal{J}(\beta_0)$ is positive definite.

These conditions are *local*; if we are allowed to assume the needed level of continuity, we may even assert that the conditions are *pointwise*. That is, to verify that a point is a local minimizer, we generally need only check the value of the first two derivatives there. Hence, these conditions are computationally feasible to check.

The optimization code uses derivative information to seek to move in a "downhill" or "descent" direction, and assumes convergence when the partial derivatives are sufficiently close to zero. As the optimization proceeds, the gradient and the estimated Hessian of each new candidate are examined, and convergence to a (local) minimizer will be declared based primarily on that evidence.

As we have said, we really want a *global minimizer* from the optimization code. By construction, in most of our test problems we will know the solution beforehand. We denote the value of the target parameters by $(\lambda^T, \alpha^T, Re^T)$, and the corresponding state variables by (u^T, v^T, p^T). The form of the cost functional then guarantees that at the target solution, the globally minimizing cost is 0. Yet we will see cases where the optimization code returns a local minimizer which is significantly far from the target solution, and with a higher value of the functional.

Supposing that there are local minimizers of the cost functional, aside from the global minimizer, it may actually be quite easy for the optimization code to return such a point. All that is required is that, during the course of the optimization, a point is computed which is close enough to the local minimizer. In that case, the gradient vector is likely to draw the optimization code even closer. A local optimizer generally lies in a "valley of attraction", that is, a small depression in the graph of the functional. Most optimization codes are very reluctant to consider points with higher functional values than their most recently accepted point. Thus, once the optimization code has fallen into the valley of attraction of a local minimizer, it is unable to climb back out, and it will converge to the local minimizer contained in the valley.

9.3 The Optimization Algorithm

Our search for the bump shape, inflow conditions, and Reynolds number which produce the closest match to a given set of flow data has been rephrased to become the search for the set of parameters β which produce the lowest value of some functional $\mathcal{J}(\beta)$. Before we consider the details of the specification of the particular functionals that we will use, we wish to make some general remarks about optimization, and the particular optimization algorithm we will employ.

Although we are trying to minimize the cost functional $\mathcal{J}(\beta)$, a great deal of useful information may be gleaned from the gradient vector $\nabla\mathcal{J}(\beta)$. One reason for this is that the negative of the gradient vector points in the direction of steepest descent of $\mathcal{J}(\beta)$. Moreover, Theorem 9.1 tells us that a necessary condition for a minimizer is that the gradient function vanish there.

If we were able to express the dependence of $\mathcal{J}$ on β explicitly, say, in a formula, then we might seek candidates for a minimizer by computing formulas for the partial derivatives of $\mathcal{J}$ with respect to each component β_i, and trying to solve for those sets of parameters β where all the partial derivatives vanish:

$$\frac{\partial\mathcal{J}(\beta)}{\partial\beta_i} = 0. \tag{9.1}$$

However, for the discrete problem, we have no formula for $\mathcal{J}$, and our only understanding of its properties comes about by explicitly evaluating $\mathcal{J}$ at a sequence of points. Therefore, the optimization is most naturally an iterative process. On the other hand, in this iterative process, we will still seek points where the gradient vanishes. The particular algorithm that we employ is a version of the *model/trust region* method.

The iterative process that we will carry out involves constructing a local model of the behavior of $\mathcal{J}$, which we expect is valid over some small "trust region" centered at the most recently accepted point. We then predict the location of a point which lies within the trust region, and at which $\mathcal{J}$ is expected to have a lower value. This prediction is based in part on the gradient direction, but uses other information as well. We then check the actual value of $\mathcal{J}$ at the predicted point. If the value is lower, as expected, then we may accept this point as the next iterate, and construct a new trust region around it. Otherwise, we retain the old iterate, shrink the trust region and try again.

In this process, gradient information is extremely valuable; not only does it tell us *when* we have reached a candidate minimizer, but it also tells us *where* to look. This is because the negative of the gradient vector points in the direction of maximum decrease of the functional. Moreover, a gradient give us information about the behavior of $\mathcal{J}$ in the entire neighborhood of a point. Many optimization algorithms require gradients of the cost function. We will not be able to supply the exact gradient, but we will be able to approximate it using discretized sensitivities or finite difference sensitivities.

At any time, the algorithm may decide to stop because it believes that the minimizer has been found. Such a decision will be based on the estimated value of the gradient, on the values of the functional at the previous points, and on the relative change in the position of the estimated location of the minimizer. The algorithm may also decide to stop because it suspects an error in the data, such as an inconsistency between the gradient data and the cost functional values. Otherwise, the algorithm will proceed to a new step, continuing to try to reduce the value of the functional.

This is as much as we need to know right now about the design and behavior of the optimization code. Further details about the algorithm are reported in Chapter 17.

9.4 Defining a Cost Functional

Our class of flow problems completely specified the shape of the region except for the bump, the boundary conditions except for the inflow, and the state equations except for the value of the Reynolds number. The quantities left unspecified, which we called the "flow parameters", may then be freely varied to produce a variety of flow problems. We assume that any set of parameter values (λ, α, Re) represents a well-defined flow problem. For convenience, we will speak as though there exists uniquely a corresponding continuous flow solution (u, v, p), although we know that the situation is actually more complicated. If we represent the parameters by β, we may emphasize the dependence of the flow solution on the parameters by writing, for instance $u(\beta)$ or $u(\lambda, \alpha, Re)$.

The desired behavior that we are seeking to optimize is for the flow solution to come as close as possible to some given flow data along a single vertical profile line in the flow region. Specifically, let us suppose that along the line $x = x_s$ we are given the following functions, or *target data*, $u^T(x_s, y)$, $v^T(x_s, y)$ and $p^T(x_s, y)$, for $0 \leq y \leq 3$. It is possible that these functions are utterly arbitrary, in which case we might expect poor approximation; it is possible that the profile data was generated as the flow solution of a discrete problem, but that this flow solution does not lie in the feasible space, so that we can, at best approximate its behavior along the profile line; finally, these functions may in fact be the profiles of an actual flow solution of a discrete problem from the feasible parameter space, generated by a set of parameters which may be designate as β^T. Only in this last case is it permissible to write $u^T(x, y) = u(x, y, \beta^T)$.

In the latter case, we might be tempted to assume that an alternate form of the optimization problem would be to recover the generating parameters β^T. We will see that this is *not* an equivalent problem. Many flows, corresponding to parameter values quite distinct from β^T, might come close to the desired behavior, and be locally optimal solutions, while corresponding to a set of parameters quite distinct from the target set. On the other hand, simply by the assumed continuity of the flow solution $u(x, y, \beta)$ as a function of β, it is correct to expect that if an optimization produces an answer very close to β^T, that the flow profiles will be close to the target profiles, and the desired behavior nearly achieved.

Now to measure how well a given flow solution approximates the target data, we want to define a *cost functional*, which we may write as $J(u, v, p, \beta)$. The cost functional should compute, in some reasonable way, the closeness between the desired and achieved behaviors. Let us suppose that in our case, we are only interested in the discrepancy for the horizontal velocity component, between the target data and the computed flow solution. A natural choice of cost functional for the continuous problem would then be:

$$J(u, v, p, \beta) = \int_{x=x_s} (u(x, y) - u^T(x_s, y))^2 \, dy. \tag{9.2}$$

Since we're actually only going to be able to solve discrete flow problems, it is necessary for us to make the obvious transformations so that we can produce a corresponding cost functional for the discrete problem, which we write $J^h(u^h, v^h, p^h, \beta)$. Since the discrete flow solution actually produces functions defined over the whole region, the cost functional for the discrete problem will formally look identical to that for the continuous case:

$$J^h(u^h, v^h, p^h, \beta) = \int_{x=x_s} (u^h(x_s, y) - u^T(x_s, y))^2 \, dy. \tag{9.3}$$

While this is the basic form for the cost functional, note that we haven't specified the actual value x_s yet. We will discuss this matter shortly. We will also reserve the right to vary the form of this cost functional by adding new terms, in our case to control undesirable oscillations in the solution.

9.5 The Role of the Cost Functional in Optimization

The cost functional allows us to specify with a single number the closeness of any given flow profile to the desired target profile. That makes it simple to define the problem: we want the "best" flow solution, one for which there's no flow solution with a lower cost functional. We also may have to settle for a "locally best" flow solution, for which there is no nearby flow solution with a lower cost functional.

Once the cost functional is specified, we can pose the *optimization problem for the parameterized continuous flow equations*:

> Suppose that, for every choice of values of the parameters β there corresponds a continuous Navier Stokes flow problem with a solution $(u(\beta), v(\beta), p(\beta))$, which we will call a feasible flow. Suppose that a cost functional $J(u, v, p, \beta)$ is defined for flow functions (u, v, p) and parameters β. The **continuous optimization problem** seeks the parameter values β^* and the corresponding flow solution $(u(\beta^*), v(\beta^*), p(\beta^*))$ such that $J(u(\beta^*), v(\beta^*), p(\beta^*), \beta^*)$ is minimized over all feasible flows $(u(\beta), v(\beta), p(\beta))$.

In analogy, we also pose the *optimization problem for the discrete parameterized flow equations*:

> Suppose that, for every choice of values of the parameters β there corresponds a discrete Navier Stokes flow problem, and that this problem has a flow solution $(u^h(\beta), v^h(\beta), p^h(\beta))$, which we will call a (discrete) feasible flow. Suppose that a cost functional $J^h(u^h, v^h, p^h, \beta)$ is defined for all flow functions (u^h, v^h, p^h) and parameters β. The **discrete optimization problem** seeks the parameter values β^* and the corresponding flow solution $(u^h(\beta^*), v^h(\beta^*), p^h(\beta^*))$ such that

$J^h(u^h(\beta^*), v^h(\beta^*), p^h(\beta^*), \beta)$ is minimized over all feasible flows $(u^h(\beta), v^h(\beta), p^h(\beta))$.

9.6 The Cost Functional as a Function of the Problem Parameters

We will find it advantageous to be able to express the optimization problem without explicit mention of the flow variables (u, v, p). It should be clear that we can do this, at least formally. If we can assume that the parameters β uniquely determine the flow solution, or that this is true at least in a local sense, as discussed in Chapter 3, then a function which depends on u and β can be replaced by an equivalent function depending only on β.

In particular, let us suppose that, instead of arbitrary flow variables (u, v, p), we are only going to evaluate the cost functional at flow variables that correspond to the argument β, so that we would write:

$$J(u(\beta), v(\beta), p(\beta), \beta) = \int_{x=x_s} (u(x, y, \beta) - u^T(x_s, y, \beta))^2 \, dy. \tag{9.4}$$

Since knowledge of β determines $u(x, y, \beta)$, we really only need to know β to evaluate J in this case. We can simply eliminate the mediation of the state variables, and regard the cost functional as a direct function of the parameter, arriving at what we will call the *constrained cost functional*:

$$\begin{aligned} \mathcal{J}(\beta) &\equiv J(u(\beta), v(\beta), p(\beta), \beta) && (9.5) \\ &= \int_{x=x_s} (u(x, y, \beta) - u^T(x_s, y))^2 \, dy. && (9.6) \end{aligned}$$

Note the distinction between the unconstrained functional J and the constrained functional $\mathcal{J}$: the cost functional J is defined for arbitrary functions u, v, p, but $\mathcal{J}$ requires the specific functions $u(\beta)$, $v(\beta)$ and $p(\beta)$ that satisfy the Navier Stokes problem defined by the parameters β. In a natural way, we define the constrained discrete cost functional $\mathcal{J}^h(\beta)$ in terms of $J^h(u^h, v^h, p^h, \beta)$.

9.7 Computing the Gradient of the Cost Functional

The optimization code will require the gradient of the cost functional. This computation would be straightforward if $\mathcal{J}(\beta)$ were given via an explicit formula. But we have no formula for the cost functional in terms of β, and we must devise a way of approximating the gradients. For any particular parameter component, we can write out the relationship between an entry of $\nabla\mathcal{J}$ and the derivatives of the original cost functional J:

$$\begin{aligned}
\frac{\partial \mathcal{J}(\beta)}{\partial \beta_i} &= \frac{DJ(u(\beta), v(\beta), p(\beta), \beta)}{D\beta_i} \\
&= \frac{\partial J(u(\beta), v(\beta), p(\beta), \beta)}{\partial u}\ u_{\beta_i} \\
&+ \frac{\partial J(u(\beta), v(\beta), p(\beta), \beta)}{\partial v}\ v_{\beta_i} \\
&+ \frac{\partial J(u(\beta), v(\beta), p(\beta), \beta)}{\partial p}\ p_{\beta_i} \\
&+ \frac{\partial J(u(\beta), v(\beta), p(\beta), \beta)}{\partial \beta_i},
\end{aligned} \tag{9.7}$$

while, for the discrete problem, we write the corresponding formula in terms of derivatives of J^h with respect to the individual finite element coefficients:

$$\begin{aligned}
\frac{\partial \mathcal{J}^h(\beta)}{\partial \beta_i} &= \frac{DJ^h(u^h(\beta), v^h(\beta), p^h(\beta), \beta)}{D\beta_i} \\
&= \sum_k \frac{\partial J^h(u^h(\beta), v^h(\beta), p^h(\beta), \beta)}{\partial u_k^h}\ (u_k^h)_{\beta_i} \\
&+ \sum_k \frac{\partial J^h(u^h(\beta), v^h(\beta), p^h(\beta), \beta)}{\partial v_k^h}\ (v_k^h)_{\beta_i} \\
&+ \sum_k \frac{\partial J^h(u^h(\beta), v^h(\beta), p^h(\beta), \beta)}{\partial p_k^h}\ (p_k^h)_{\beta_i} \\
&+ \frac{\partial J^h(u^h(\beta), v^h(\beta), p^h(\beta), \beta)}{\partial \beta_i}.
\end{aligned} \tag{9.8}$$

These formulas require the values of the continuous or discrete sensitivities. These are unobtainable for the continuous case, and difficult to calculate in the discrete case. Therefore, we consider ways of estimating the discrete cost gradient, using approximations to the discrete sensitivities.

If we have computed the discretized sensitivities, for instance, we may try the approximation:

$$\begin{aligned}
\frac{\partial \mathcal{J}^h(\beta)}{\partial \beta_i} \approx (\frac{\partial \mathcal{J}(\beta)}{\partial \beta_i})^h \quad &\equiv \quad \sum_k \frac{\partial J^h(u^h(\beta), v^h(\beta), p^h(\beta), \beta)}{\partial u_k^h} \; (u_{k,\beta_i})^h \\
&+ \quad \sum_k \frac{\partial J^h(u^h(\beta), v^h(\beta), p^h(\beta), \beta)}{\partial v_k^h} \; (v_{k,\beta_i})^h \\
&+ \quad \sum_k \frac{\partial J^h(u^h(\beta), v^h(\beta), p^h(\beta), \beta)}{\partial p_k^h} \; (p_{k,\beta_i})^h \\
&+ \quad \frac{\partial J^h(u^h(\beta), v^h(\beta), p^h(\beta), \beta)}{\partial \beta_i}. \qquad (9.9)
\end{aligned}$$

Here, we have had to write awkward quantities such as

$$(u_{k,\beta_i})^h \qquad (9.10)$$

to stand for the k-th finite element coefficient of the discretized velocity sensitivity with respect to β_i, in other words, the k-th coefficient of $(u_{\beta_i})^h$.

If we have computed the finite coefficient differences, then we can instead use the approximation:

$$\begin{aligned}
\frac{\partial \mathcal{J}^h(\beta)}{\partial \beta_i} \quad &\approx \quad \sum_k \frac{\partial J^h(u^h(\beta), v^h(\beta), p^h(\beta), \beta)}{\partial u_k^h} \; \frac{\Delta u_k^h}{\Delta \beta_i} \\
&+ \quad \sum_k \frac{\partial J^h(u^h(\beta), v^h(\beta), p^h(\beta), \beta)}{\partial v_k^h} \; \frac{\Delta v_k^h}{\Delta \beta_i} \\
&+ \quad \sum_k \frac{\partial J^h(u^h(\beta), v^h(\beta), p^h(\beta), \beta)}{\partial p_k^h} \; \frac{\Delta p_k^h}{\Delta \beta_i} \\
&+ \quad \frac{\partial J^h(u^h(\beta), v^h(\beta), p^h(\beta), \beta)}{\partial \beta_i}. \qquad (9.11)
\end{aligned}$$

Finally, we could choose to approximate the gradient directly, using the formula:

$$\frac{d\mathcal{J}^h(\beta)}{d\beta_i} \approx \frac{\Delta\mathcal{J}^h(\beta)}{\Delta\beta_i} \equiv \frac{\mathcal{J}^h(\beta+\Delta\beta_i)-\mathcal{J}^h(\beta)}{\Delta\beta_i}. \tag{9.12}$$

Assuming we use either the discretized sensitivities or finite coefficient differences, it remains to compute the partial derivatives of J with respect to u_k, v_k, and p_k. This is usually not a problem, since J is generally presented as a simple, explicit formula in terms of these variables. The partial derivative with respect to β_i should also be easy to compute, as long as there are no secondary effects on the cost functional due to changes in the region's geometry. That might happen, for instance, if the profile line were placed in a region which was affected by changes to β. As it happens, the profile line will always be placed in the fixed portion of the region, so we will not have to concern ourselves with such effects.

Note that, if we use the finite coefficient difference approach, the computation of component i of the gradient will require that we carry out a full flow solve to find $u^h(\beta+\Delta\beta_i)$, $v^h(\beta+\Delta\beta_i)$ and $p^h(\beta+\Delta\beta_i)$. Moreover, we must choose a "reasonable" increment β_i that is small enough for an accurate approximation of the partial derivative while being large enough to avoid roundoff problems. Finally, as we have already noted, at least a few nodes *must* move if we are using a shape parameter; perhaps most of the nodes move. If we actually wish to compute sensitivities at a *fixed* spatial location, then the finite coefficient differences computed at moving nodes must be adjusted as described earlier to subtract off changes in the flow solution that are merely due to spatial displacement of the nodes.

If we use the discretized sensitivity approach, then we will generally just have finished a Newton iteration for the flow solution, and so will have at hand exactly the matrix we need for the sensitivity system, already assembled and factored. Therefore, the only cost will be in the assembly of the right hand sides for each sensitivity, and the solution of the already factored linear system. We will see that, in general, this approach can be actually cheap compared to the finite coefficient difference method. Instead of $NPar$ extra flow solutions, we simply have $NPar$ linear systems to solve. The main drawback to using the discretized sensitivities for geometric parameters is their inherent error. Once we have chosen a particular finite element mesh, there is an automatic discrepancy between the discretized and true sensitivities; the only sure cure for this is to refine the finite element mesh, a costly step.

9.8 The Formula for the Hessian

Although we will not do so here, we wish to note that, if we had approximated the *second order sensitivities* using discretized sensitivities or finite coefficient differences, the Hessian matrix could also be supplied to the optimization algorithm. We exhibit the appropriate formula for the continuous case, although for simplicity we suppress the terms involving differentiation with respect to the variables v and p:

$$\begin{aligned}
\frac{\partial^2 \mathcal{J}(\beta)}{\partial\beta_i \partial\beta_j} &= \frac{D^2 J(u(\beta), v(\beta), p(\beta), \beta)}{D\beta_i D\beta_j} \\
&= \frac{\partial^2 J(u(\beta), v(\beta), p(\beta), \beta)}{\partial u^2(\beta)} \frac{\partial u(\beta)}{\partial \beta_i} \frac{\partial u(\beta)}{\partial \beta_j} \\
&+ \frac{\partial^2 J(u(\beta), v(\beta), p(\beta), \beta)}{\partial u \partial \beta_j} \frac{\partial u(\beta)}{\partial \beta_i} \\
&+ \frac{\partial^2 J(u(\beta), v(\beta), p(\beta), \beta)}{\partial u \partial \beta_i} \frac{\partial u(\beta)}{\partial \beta_j} \\
&+ \frac{\partial J(u(\beta), v(\beta), p(\beta), \beta)}{\partial u} \frac{\partial^2 u(\beta)}{\partial \beta_i \partial \beta_j} \\
&+ \frac{\partial^2 J(u(\beta), v(\beta), p(\beta), \beta)}{\partial \beta_i \partial \beta_j}. \qquad (9.13)
\end{aligned}$$

9.9 Optimization With Approximate Gradients

Whether we use the chain rule on discretized sensitivities or finite coefficient differences, or use finite cost functional differences directly, we will be producing an approximation of the cost gradient. To halve the approximation error in a first order finite difference quotient, we have only to halve the size of the perturbation, such as $\Delta\alpha$. A new estimate then costs us one more flow solve. But, if our discretized sensitivity error is of order $O(h)$, to halve the approximation error in a discretized sensitivity, we must halve h, which means we roughly quadruple the number of nodes and unknowns, making the work of factoring the Jacobian increase by a factor of more than 16. Thus there is a limit to how much we can reduce h simply to improve the gradient estimates.

The effect of using an inaccurate gradient depends greatly on which particular algorithm is being used. For a variation of the model trust region optimization method, Carter [8] showed that convergence to a minimizer would still be achieved as long as the approximate gradient $(\nabla\mathcal{J}(\beta))^h$ had a sufficiently positive projection on the correct gradient $\nabla\mathcal{J}^h(\beta)$. In such a case, the negative of the approximate gradient would still define a descent direction for the function to be minimized, though generally not a *steepest* descent direction. The condition given is equivalent to the requirement that there exist some fixed positive γ, so that for all parameter values β at which $(\nabla\mathcal{J}(\beta))^h$ does not vanish, it is true that:

$$0 < \gamma\ (\nabla\mathcal{J}(\beta))^h \cdot (\nabla\mathcal{J}(\beta))^h \leq (\nabla\mathcal{J}(\beta))^h \cdot \nabla\mathcal{J}^h(\beta). \tag{9.14}$$

While we don't have the exact gradients, we may carry out an approximate version of this check on the discretized sensitivity gradients, using finite difference gradient estimates.

9.10 Example: A Cost Functional Using Horizontal Velocities Only

As an example of a cost functional, let us consider a case where we are given the value of the function $u^T(x_s, y)$, with the profile line at $x_s = 3$:

$$J_2(u, v, p, \beta) = \int_{x_s=3} (u(x_s, y) - u^T(x_s, y))^2 \, dy. \tag{9.15}$$

which we may rewrite in terms of β as:

$$\mathcal{J}_2(\beta) = \int_{x_s=3} (u(x_s, y, \beta) - u^T(x_s, y))^2 \, dy. \tag{9.16}$$

Now, we would like to compute a partial derivative of the form $\dfrac{\partial J_2}{\partial u}$ but, as in the calculation of the FP operator in Chapter 3, we must use the method of variations in order that the computation makes sense. In other words, we use the fact that:

$$\frac{\partial J_2}{\partial u}(u_0, v_0, p_0, \beta_0)\ \tilde{u} = \lim_{\epsilon \longrightarrow 0} \frac{J_2(u_0 + \epsilon\tilde{u}, v_0, p_0, \beta_0) - J_2(u_0, v_0, p_0, \beta_0)}{\epsilon} \tag{9.17}$$

to compute that

$$\begin{aligned}
& \frac{\partial J_2}{\partial u}(u_0, v_0, p_0, \beta_0)\ \tilde{u} && (9.18)\\
= \ & \lim_{\epsilon \longrightarrow 0} \frac{\int_{x_s=3} ((u + \epsilon\tilde{u})(x_s, y) - u^T(x_s, y))^2 \, dy - \int_{x_s=3} (u(x_s, y) - u^T(x_s, y))^2 \, dy}{\epsilon} && \\
& && (9.19)\\
= \ & \lim_{\epsilon \longrightarrow 0} \frac{\int_{x_s=3} ((u + \epsilon\tilde{u})(x_s, y) - u^T(x_s, y))^2 - (u(x_s, y) - u^T(x_s, y))^2 \, dy}{\epsilon} && (9.20)\\
= \ & \lim_{\epsilon \longrightarrow 0} \frac{\int_{x_s=3} 2\epsilon u(x_s, y)\tilde{u}(x_s, y) - 2\epsilon\tilde{u}(x_s, y)u^T(x_s, y) - \epsilon^2\tilde{u}^2(x_s, y) \, dy}{\epsilon} && (9.21)\\
= \ & 2\int_{x_s=3} (u(x_s, y) - u^T(x_s, y))\tilde{u} \, dy. && (9.22)
\end{aligned}$$

Now if we simply choose $\tilde{u}$ to be $\dfrac{\partial u}{\partial \beta}$, we have

$$\begin{aligned}
\frac{\partial \mathcal{J}_2(\beta)}{\partial \beta} &= \frac{\partial J_2(u(\beta), v(\beta), p(\beta), \beta)}{\partial u}\ \frac{\partial u(\beta)}{\partial \beta} && (9.23)\\
&= 2\int_{x_s=3} (u(x_s, y, \beta) - u^T(x_s, y))\frac{\partial u(\beta)}{\partial \beta}\ dy. && (9.24)
\end{aligned}$$

For the discrete problem, we can easily derive the corresponding formula:

$$
\begin{aligned}
\frac{\partial \mathcal{J}_2^h(\beta)}{\partial \beta} &= \frac{\partial J_2(u^h(\beta), v^h(\beta), p^h(\beta), \beta)}{\partial u^h} \frac{\partial u^h}{\partial \beta} && (9.25)\\
&= 2 \int_{x_s=3} (u^h(x_s, y, \beta) - u^T(x_s, y)) \frac{\partial u^h}{\partial \beta}\, dy && (9.26)\\
&= 2 \int_{x_s=3} (u^h(x_s, y, \beta) - u^T(x_s, y)) u^h_\beta\, dy. && (9.27)
\end{aligned}
$$

We can approximate this equation by using the discretized sensitivities, giving us the following computable approximation to the gradient:

$$
\begin{aligned}
\frac{\partial \mathcal{J}_2^h(\beta)}{\partial \beta} &\approx (\frac{\partial \mathcal{J}_2(\beta)}{\partial \beta})^h && (9.28)\\
&\equiv \frac{\partial J_2(u^h(\beta), v^h(\beta), p^h(\beta), \beta)}{\partial u^h}\ (u_\beta)^h && (9.29)\\
&= 2 \int_{x_s=3} (u^h(x_s, y, \beta) - u^T(x_s, y)) (u_\beta)^h\, dy. && (9.30)
\end{aligned}
$$

When the discretized target flow u^T is actually a feasible flow, that is, $u^T(x,y) = u^h(x,y,\beta^T)$ for some set of parameters β^T, this formula implies that the approximate cost gradient computed using discretized sensitivities will always be identically zero at the global minimizer $u^h(x,y,\beta^T)$.

This is *not* because of some special property of discretized sensitivities, but rather because of the factor of $u^h(x_s,y,\beta) - u^T(x_s,y)$ in the cost gradient computation; an arbitrary approximation for u^h_β would satisfy the same condition. This means that while we should expect the approximated cost gradients to be zero at a feasible target, we should not take that as evidence of special accuracy!

Now we have available three approximations to derivative quantities like $\frac{\partial u^h}{\partial \beta}$, namely the discretized sensitivities and the unadjusted and adjusted finite coefficient differences. We

Table 9.1: Cost gradients via discretized sensitivities and finite differences. The comparison is made at several sets of parameter values.

Gradient Component	Finite Coef Difference (FCD)	Adjusted FCD (AFCD)	Direct FD (DFD)	Discretized Sensitivity (DS)
		$Re = 5.0$		
$\frac{\partial \mathcal{J}_2^h}{\partial \lambda}$	-0.86784	-0.86784	-0.86784	-0.86784
$\frac{\partial \mathcal{J}_2^h}{\partial \alpha}$	-0.031580	-0.031580	-0.031580	-0.034021
$\frac{\partial \mathcal{J}_2^h}{\partial Re}$	-0.00010525	-0.00010525	-0.00010525	-0.00010527
		$Re = 7.5$		
$\frac{\partial \mathcal{J}_2^h}{\partial \lambda}$	-0.45423	-0.45423	-0.45423	-0.45423
$\frac{\partial \mathcal{J}_2^h}{\partial \alpha}$	-0.032849	-0.032849	-0.032849	-0.036069
$\frac{\partial \mathcal{J}_2^h}{\partial Re}$	-0.00017476	-0.00017476	-0.00017476	-0.00017476
		$Re = 10.0$		
$\frac{\partial \mathcal{J}_2^h}{\partial \lambda}$	-0.334E-15	-0.334E-15	0.283E-06	-0.334E-15
$\frac{\partial \mathcal{J}_2^h}{\partial \alpha}$	0.608E-15	0.608E-15	0.170E-07	0.574E-15
$\frac{\partial \mathcal{J}_2^h}{\partial Re}$	0.151E-18	0.151E-18	0.356E-11	0.151E-18

may also approximate the cost gradient directly, without using an estimate for the sensitivities, using a direct finite difference approach. In Table 9.1, we compare the cost gradients we get via these methods. The problem being solved uses three parameters, one inflow, one bump, and Re, with a mesh of 240 elements and a mesh parameter $h = 0.5$. The target data was generated previously on the same mesh, setting $(\lambda^T, \alpha^T, Re^T)$ to the values $(0.5, 0.5, 10.0)$. The program generated flow solutions and cost gradients for the parameter values $(0.25, 0.25, 5.0)$, $(0.375, 0.375, 7.5)$, and $(0.5, 0.5, 10.0)$.

Note that, at $Re = 10$, where the cost gradients should be zero, it is the direct finite difference estimate that is most out of line. This error is to be expected, of course. We said that the direct finite difference estimate gave an answer that was equal to the true gradient, plus an error of order $\Delta\beta$. Since the true gradient is zero, we are seeing exactly the error term.

All four gradient estimates agree almost completely for the first and third components of the gradient. However, there is a roughly 10% error in the estimates for the shape gradient when using discretized sensitivities. It is hard to judge this fact. We are using a crude mesh, and a finer one would improve the agreement, as we have seen in Chapter 8. Since the discretized sensitivities are arrived at by a "shortcut", we must expect to pay a price for their ease of computation. But whether these inaccuracies in the gradient will actually hamper us in the optimization efforts is a question we will have to face shortly, in Chapter 11.

9.11 Regularization of Cost Functionals

We will shortly see a number of situations in which the optimization algorithm is unable to produce a satisfactory answer for us. This is not remarkable when we consider the constraints on the optimization code. It generally uses very local information, so that it cannot "see" that just beyond a rise in the functional lies a very substantial drop in its value. The optimization code will refuse to accept a new iterate which causes the functional value to increase. Hence, once an optimization code has entered a "valley" it generally cannot escape. It is very common for a functional to have multiple local minima, and many of these local minima can have functional values that are quite large relative to the value obtained at the global minimum.

Our first question then is, if the optimization code has stumbled into a local minimum which is unsatisfactory, but for which the "valley" is reasonably shallow, can we "escape" from somehow, and force the optimization code to try to find a better minimum?

A second, milder problem can also occur. Even though a function may have only one minimizer, it may be unusually flat, or be poorly scaled, or have level sets that are very twisted. In all these cases, it can be extremely inefficient to carry out the typical optimization algorithm, which uses local directional data, takes a large step in a descent direction, and then penalizes itself severely if the step caused the functional to increase.

Finally, we may find that the minimizer uncovered by the optimization code has some undesirable property that we wish to avoid. For instance, if we are seeking the shape of a bump that will produce a given flow downstream, we may not realize that we want a smooth bump until the optimization code returns with a set of parameters that define an extremely oscillatory shape. The undesired oscillations can be controlled by adding a *penalty function*, which measures the amount of oscillation in the bump, and adds some small multiple of this amount to the cost function. A simple example might be

$$\mathcal{J}_3(\lambda, \alpha) \equiv \int_{x=x_s} (u(x,y) - u^T(x_s, y))^2 \, dy + \epsilon \int (\frac{\partial Bump(x, \alpha)}{\partial x})^2 \, dx. \tag{9.31}$$

Using such a penalty function requires an *ad hoc* choice for the quantity ϵ, and also means that when the optimization code finds the solution that minimizes J_3, there may be other flow solutions which have a smaller discrepancy integral, and hence are "better" in the old sense.

Chapter 10

A POORLY SCALED OPTIMIZATION

10.1 Introduction

In this chapter, we consider a problem which can not be accurately optimized because the cost functional has been poorly chosen. We discuss the behavior of the optimization package, and the reasons for its failure. We examine the computed cost gradient and question its accuracy. We then look closely at the velocity sensitivity field, and see clear reasons for the failure of the optimization. We then propose a simple cure for the problem.

10.2 A Poor Optimization Using Discretized Sensitivities

We consider one of our standard problems, with four parameters: a single inflow parameter λ, and three parameters $\alpha = (\alpha_1, \alpha_2, \alpha_3)$ which determine the shape of the bump. The Reynolds number Re is fixed at 1.

We set a profile line at $x_s = 9$, quite far down the channel from the bump. We then set the target parameter values $\lambda^T = 0.500$ and $\alpha^T = (0.375, 0.5, 0.375)$, and compute the target

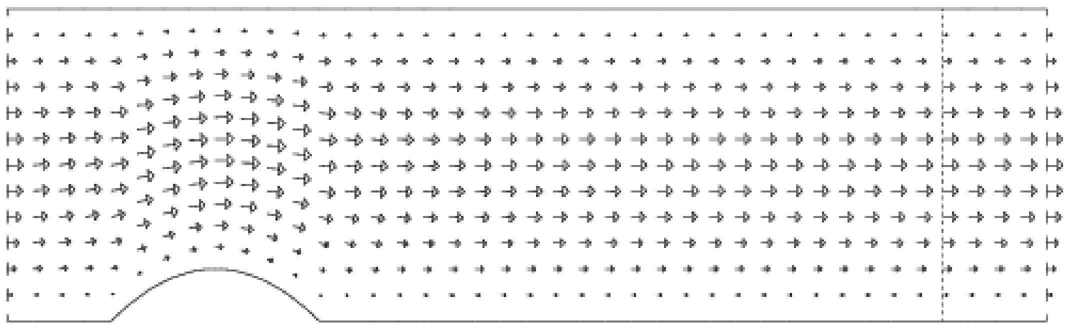

Figure 10.1: The flow region and velocity field for the target parameters. Note the profile line, at $x_s = 9$.

flow solution:

$$(u^T, v^T, p^T) = (u(\lambda^T, \alpha^T), v(\lambda^T, \alpha^T), p(\lambda^T, \alpha^T)). \tag{10.1}$$

which is displayed in Figure 10.1.

We may now define the cost functional J_1^h:

$$J_1^h(u^h, v^h, p^h, \lambda, \alpha) = \int_{x_s=9} (u^h(x_s, y) - u^T(x_s, y))^2 \, dy, \tag{10.2}$$

and in the usual way, define $\mathcal{J}_1^h$ to be J_1^h restricted to flow functions that are consistent with the Navier Stokes equations as determined by the parameters:

$$\begin{aligned} \mathcal{J}_1^h(\lambda, \alpha) &\equiv J_1^h(u^h(\lambda, \alpha), v^h(\lambda, \alpha), p^h(\lambda, \alpha), \lambda, \alpha) & (10.3)\\ &= \int_{x_s=9} (u^h(x_s, y, \lambda, \alpha) - u^T(x_s, y))^2 \, dy. & (10.4) \end{aligned}$$

We now face the problem of finding parameter values λ and α which minimize $\mathcal{J}_1^h$. We use a discretization of the flow problem with mesh parameter $h = 0.25$. We begin the optimization attempt with λ and α set to zero. Initially, we set the optimization error tolerance, called **TolOpt**, to $1.0E - 09$. When the optimization code requests a value of the gradient of $\mathcal{J}_1^h(\lambda, \alpha)$, we compute an approximation of the answer by computing the discretized sensi-

Table 10.1: Optimization results for $\mathcal{J}_1^h$, using discretized sensitivities. The third optimization did not converge after 100 steps.

TolOpt	Steps	λ	α_1	α_2	α_3	$\mathcal{J}_1^h$	$\|\|(\nabla\mathcal{J}_1)^h\|\|_\infty$
1.0E-09	20	0.499998	0.0767	0.0656	0.276	0.572E-09	0.3E-05
1.0E-10	23	0.500000	0.1257	0.1075	0.453	0.375E-10	0.3E-05
1.0E-11	100	0.500001	0.1257	0.1075	0.453	0.358E-10	0.8E-09

tivities $((u_\lambda)^h, (v_\lambda)^h, (p_\lambda)^h)$ and $((u_\alpha)^h, (v_\alpha)^h, (p_\alpha)^h)$ and applying Equation (9.9) to compute an approximation to the desired gradient, which we denote $(\nabla\mathcal{J}_1(\lambda, \alpha))^h$.

After 20 steps, the optimization code reports satisfactory convergence. However, the parameters that the optimization code returns are $\lambda = 0.499998$, $\alpha = (0.0767, 0.0656, 0.276)$. These values seem unreasonably far from the target solution. A comparison of Figures 10.2 and 10.3 confirms that the difference between the shapes of the target and computed bumps is considerable. Despite this fact, the cost functional, which started out at $\mathcal{J}_1^h(\lambda, \alpha) = 0.400$, has dropped to $0.572E - 09$, very close to the minimum value of 0.

Seeking a set of parameters that are closer to the correct values, we tried lowering the tolerance **TolOpt**. The first reduction in the tolerance produces a "better" result in just 23 steps; the new bump has risen closer to its maximum height of 0.5, but the peak occurs at the *end* of the bump rather than at the middle. If we reduce the tolerance by a factor of 10 again, the optimization code fails to converge after 100 steps, and produces results almost identical to the previous set. The results are summarized in Table 10.1, where we list the optimization tolerance, the number of steps, the final point of the optimization, the cost functional evaluated at that point, and the maximum norm of the (approximate) gradient of the cost functional. It is clear that this problem is beyond the abilities of the optimization code to correct, and that we may need to look elsewhere for a remedy.

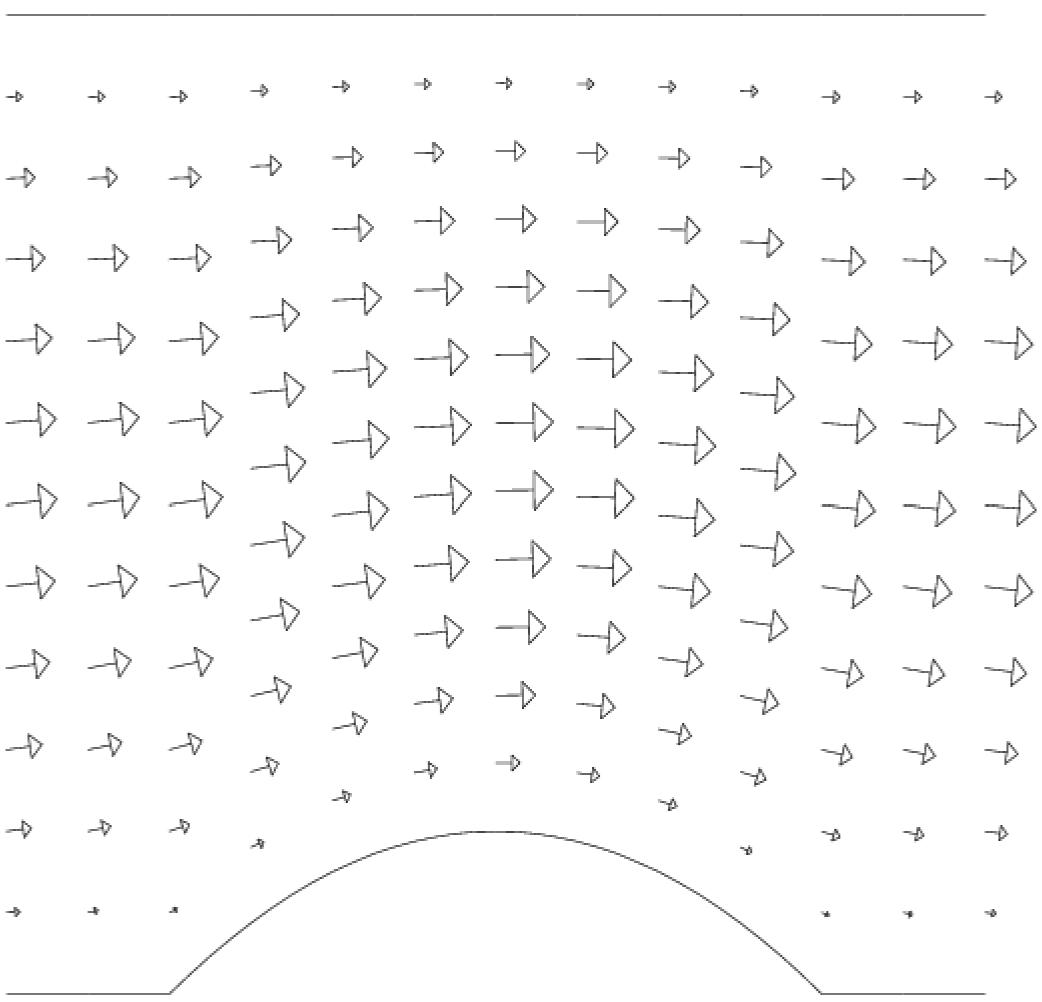

Figure 10.2: The bump and velocity field for the target parameters. This figure shows the portion of the flow region near the bump.

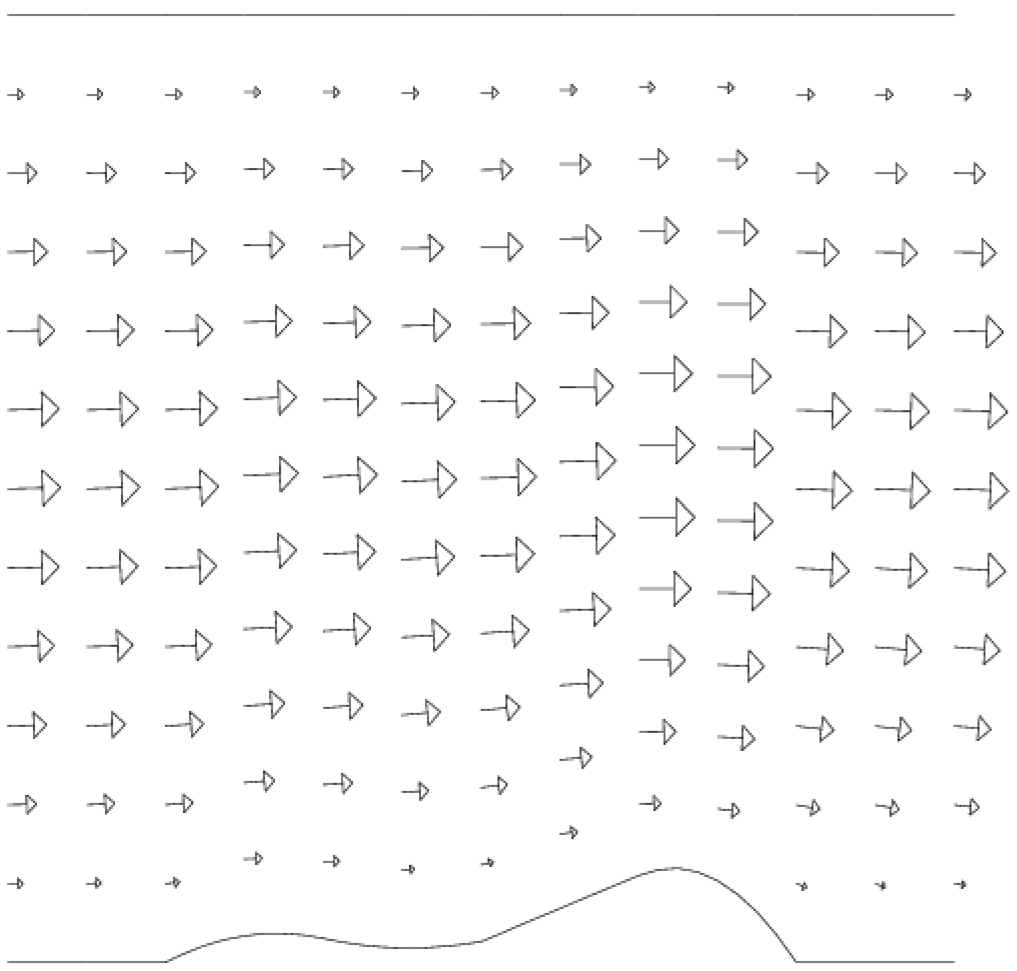

Figure 10.3: The computed bump and velocity field using discretized sensitivities. This is the solution computed for **TolOpt**=1.0E-9.

10.3 Approximating the Cost Gradient with Finite Differences

It is reasonable to assume that the optimization algorithm failed because of inconsistent data. In other words, we had asked the optimization algorithm to seek minimizers of the cost functional, and we provided it with an approximation to the gradient of the cost functional. However, especially near the end of an optimization, errors in the gradient can delay or defeat further progress.

We can investigate this possibility, since we have an alternative, and presumably more accurate, approximation of the cost gradient, via finite cost gradient differences. That is, once we had a flow solution (u, v, p) for a particular set of parameters β, and the corresponding cost $\mathcal{J}_1^h(\beta)$, we perturbed each component of β in turn, solved the corresponding flow problem, evaluated the cost of the solution, and used the formula:

$$\frac{\partial \mathcal{J}_1^h(\beta)}{\partial \beta_i} \approx \frac{\mathcal{J}_1^h(\beta + \Delta\beta_i) - \mathcal{J}_1^h(\beta)}{\Delta\beta_i}. \tag{10.5}$$

Using this modification, we again solved the flow optimization problem with **TolOpt** set to $1.0E-09$. The optimization results were almost ludicrous: while the inflow parameter was well approximated, the bump parameters were all identically zero! We repeated the effort three times, with successively smaller optimization tolerances. None of these efforts produced any significant improvement in the bump parameters. The results are summarized in Table 10.2.

Unlike the previous optimization, this time the optimization code reported unsatisfactory convergence behavior, of the type "false convergence". This message may be interpreted to mean that the optimization code believes there is a discrepancy between the cost functional and the gradient. This is quite a surprise, since we specifically sought to improve the gradient calculation. However, as will become plain later, this particular problem is especially

Table 10.2: Results for $\mathcal{J}_1^h$, using finite cost gradient differences.

TolOpt	Steps	λ	α_1	α_2	α_3	$\mathcal{J}_1^h$	$\|\|\nabla\mathcal{J}_1^h\|\|_\infty$
1.0E-09	10	0.499996	0.0	0.0	0.0	0.286E-08	0.2E-06
1.0E-10	13	0.499996	0.0	0.0	0.0	0.286E-08	0.2E-06
1.0E-11	16	0.499996	0.9E-12	0.8E-12	0.3E-11	0.286E-08	0.2E-06
1.0E-12	20	0.499999	0.2E-11	0.1E-11	0.7E-11	0.286E-08	0.2E-06

unsuitable for such finite difference estimates.

10.4 Approximating the Sensitivities with Finite Differences

As a final approach to solving this problem, we tried going back to the chain rule calculation of the cost gradients, Equation (9.8). However, this time, we tried to approximate the sensitivities for the discrete problem by finite coefficient differences. That is, once we had computed each of the finite element coefficients $u_k^h(\beta)$, we computed $u_k^h(\beta+\Delta\beta)$, and made the approximation:

$$(u_k^h)_\beta(\beta) \approx \frac{(u_k^h)(\beta+\Delta\beta) - u_k^h(\beta)}{\Delta\beta}. \tag{10.6}$$

We already have discussed the fact that this finite difference approximation to $\dfrac{\partial u^h}{\partial \beta_i}$ has a built-in inaccuracy whenever moving nodes are involved. Nonetheless, we wished to compare this approach with the use of discretized sensitivities.

A glance at Figure 10.5 will quickly confirm that the bump corresponding to the parameter values we have achieved looks nothing like the bump for the target data. And yet the cost functional is quite low, so that the optimization algorithm seems to have done part of its job; it has decreased the cost functional, though not enough.

But how can the cost functional be so low, while the bump parameters are so far from being correct? Perhaps it is the specification of the cost functional itself that is at fault. The fact

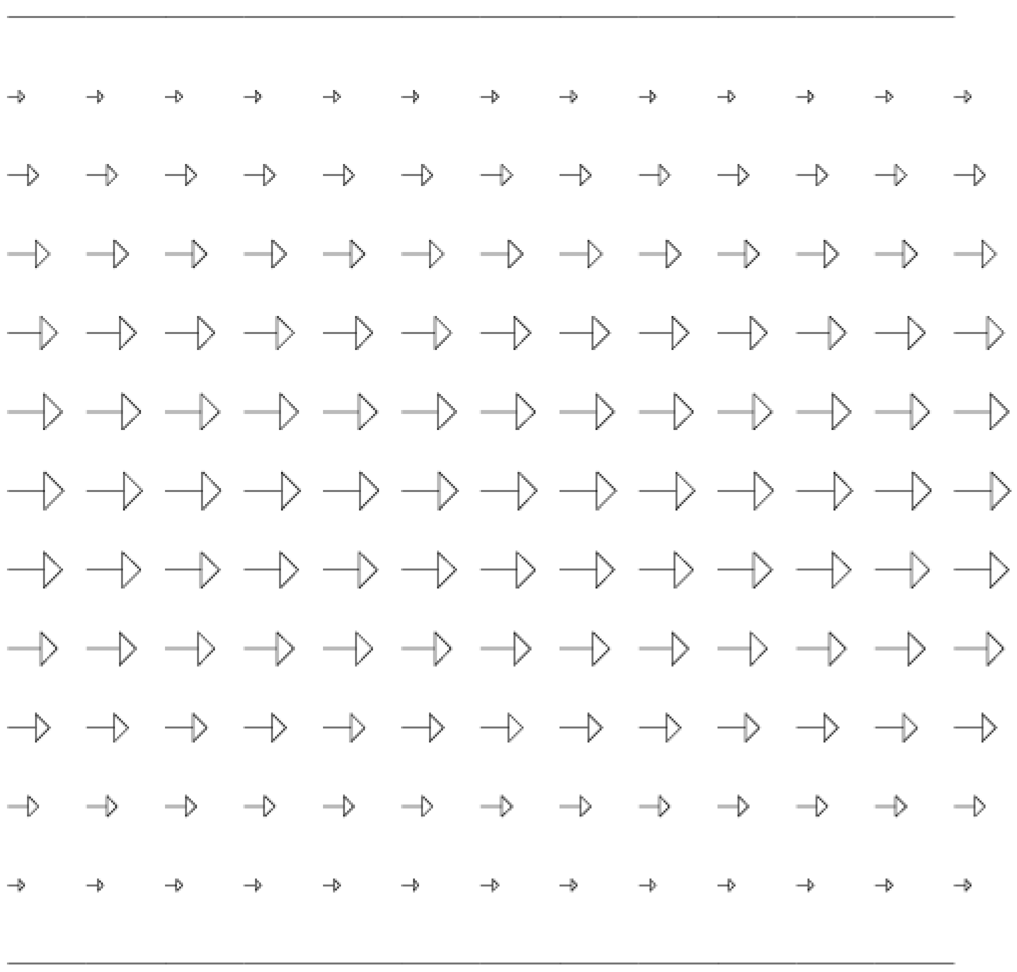

Figure 10.4: The computed bump and velocity field using a finite difference cost gradient.

Table 10.3: Results for $\mathcal{J}_1^h$, using finite coefficient differences.

TolOpt	Steps	λ	α_1	α_2	α_3	$\mathcal{J}_1^h$	$\|\|\nabla \mathcal{J}_1^h\|\|_\infty$
1.0E-09	19	0.499998	0.084	0.074	0.316	0.321E-09	0.4E-05
1.0E-10	45	0.500004	0.114	0.100	0.426	0.155E-10	0.8E-05
1.0E-11	46	0.500001	0.112	0.098	0.419	0.334E-11	0.4E-08
1.0E-12	40	0.500001	0.112	0.099	0.422	0.318E-11	0.1E-09

that the computed optimizing bump is so dramatically far off from the correct one suggests to us that $\mathcal{J}_1^h$ is not a very good measure of closeness.

10.5 Investigating the Poor Convergence

We have tended to judge the optimization by how well it "rediscovers" the parameter values β^T that were used to define the target data $u^T(x, y)$. This is a natural expectation, but it is not, strictly speaking, proper.

The optimization code's task is simply to produce a set of parameters β that are "locally optimal" to within the specified tolerance **TolOpt**. This means, roughly, that the parameters β produce a value of $\mathcal{J}_1^h$ that is believed to be very near to the minimum value in some small neighborhood. In other words, no "large" decrease in $\mathcal{J}_1^h$ can be expected by taking a small step away from the estimated solution β.

Can we convince ourselves that the optimization code has actually performed its task well, despite our distaste for the results? To believe that this is so, we must show that no small step away from the estimated minimizer produces a large decrease in the cost. But this is quite easy to check. For instance, consider the very first solution we got, using discretized sensitivities. It lies at a Euclidean distance of about 0.53 units from the true solution, in parameter space. Over that distance, the cost functional decreases from 0.572E-09 to 0. If we suppose that in the neighborhood of the computed and correct minimizers, the cost

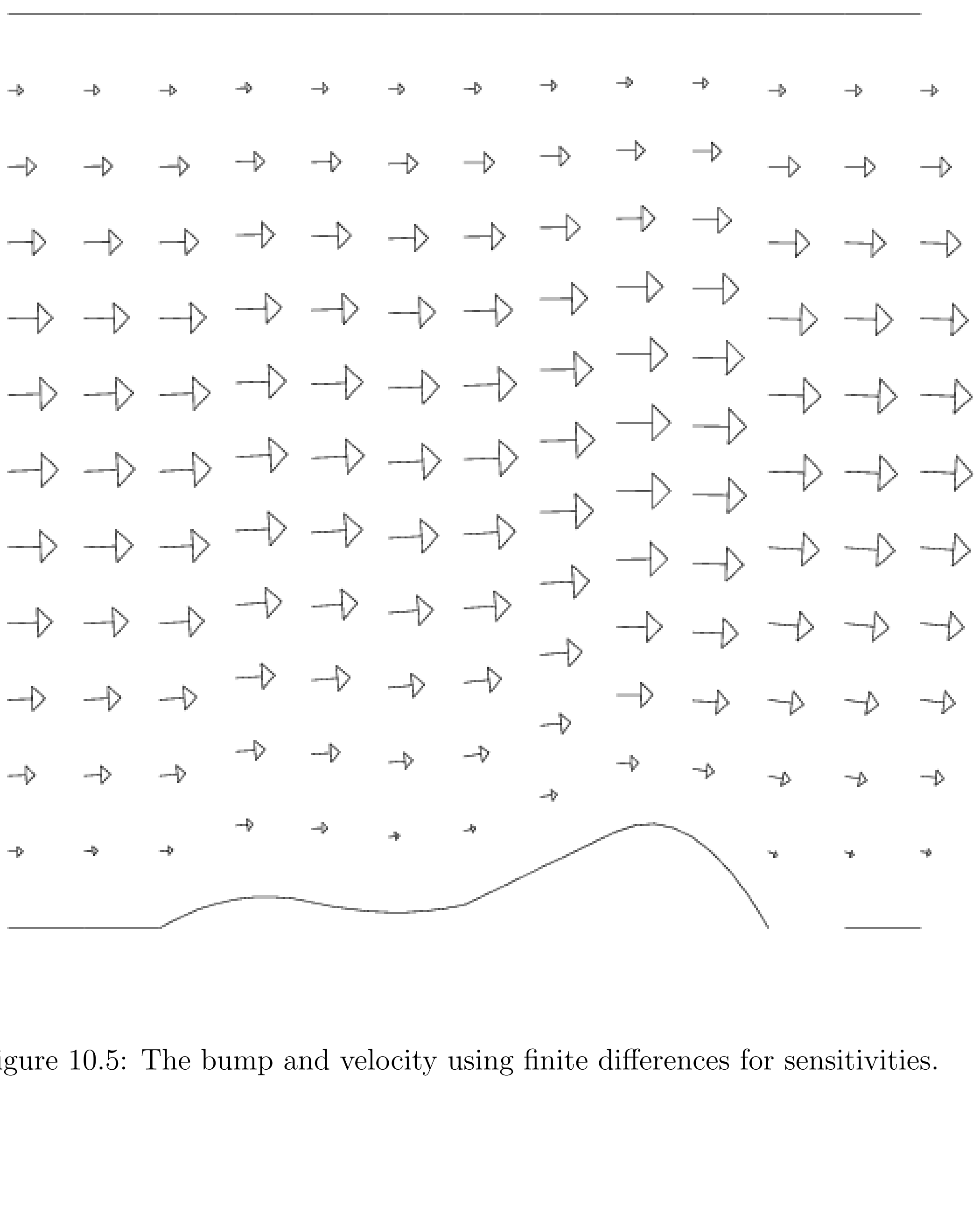

Figure 10.5: The bump and velocity using finite differences for sensitivities.

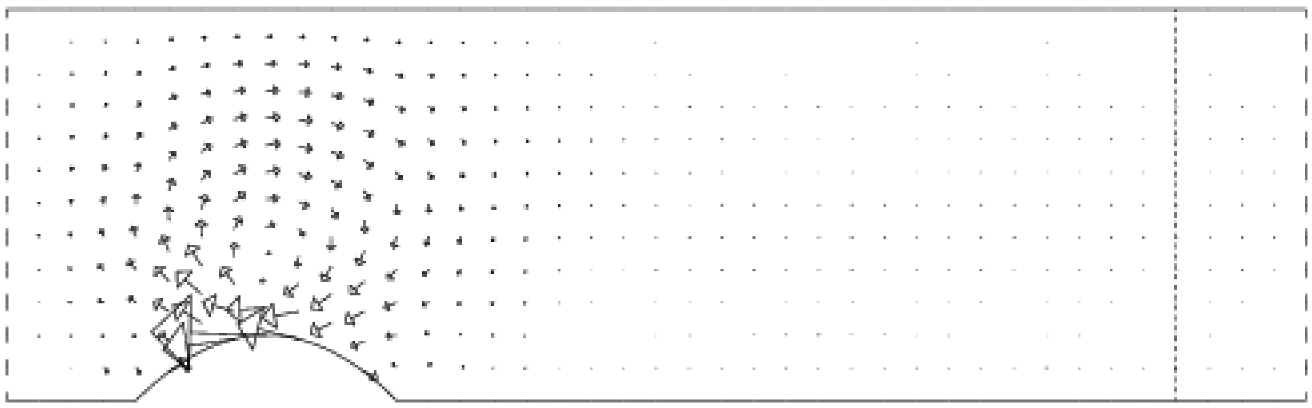

Figure 10.6: Discretized velocity α_2-sensitivity field.
The profile line is shown at $x_s = 9$.

functional does not vary significantly from these two values, then it is quite reasonable for the optimization code to assume that it has reached a minimizer (since the gradients must be extremely small) and that no "significantly" better minimizer lies close to the computed one. In fact, given such small cost functional values, it would not be implausible to imagine that there is actually a slight local minimum at the computed point. These remarks apply to the points returned in the other computations as well, even to the displeasing results of using finite difference cost gradients.

This is not a satisfactory state of affairs. The values of the cost functional seem to be unusually small, even for parameters that are quite far from the target values. This makes for a very difficult optimization indeed. It suggests that the cost functional is relatively *insensitive* to changes in the parameters. It would be tempting to assume that therefore the flow solution (u^h, v^h, p^h) is also insensitive to parameter changes, but this is not correct, as is made plain when we plot the discretized sensitivities over the flow region, in Figure 10.6.

The figure makes the problem clear: the flow solution *as a whole* is not insensitive to parameter changes, but the portion of the solution along the profile line $x_s = 9$ is indeed very insensitive. This is actually a feature of low Reynolds number flow through a channel; even when disrupted by the bump, the flow has a very strong tendency to return to Poiseuille flow once it is again passing through a rectangular region. Thus, by the time the flow has reached the profile line, almost all traces of the influence of the bump have been overwhelmed.

Once we understand the meaning of this sensitivity plot, a reasonable solution is clear. We don't want to try to change the physics of the flow, but we are free to decide where we make measurements (and for that matter, *what* we measure there). If we simply move the profile line from $x_s = 9$ to some point much closer to the bump, then changes in the parameters will have a much larger effect on the portion of the flow solution that is sampled. This, in turn, will make the cost functional values larger, and much less flat. And this, finally, will make the optimization problem significantly easier to handle.

In fact, for all further computations, we will move the profile line right up to the back of the bump, setting $x_s = 3$. In the next chapter, we will begin using this new cost functional. Although the convergence behavior will be much improved, we will discover a new complication!

Chapter 11

A DISCRETIZED SENSITIVITY FAILURE

11.1 Introduction

Having learned from the experiences with the profile line at $x_s = 9$, we now move the line much closer to the bump, and attempt to optimize the new problem.

We find that the optimization code returns a solution which is not the global minimizer, that is, the target parameters we used to set up the problem. Moreover, the returned solution does not even seem to be a local minimizer; the cost gradient entries are far too large. It is necessary that we understand what has happened before proceeding.

After a brief consideration of the numerical evidence, we turn to graphical analysis. We plot the cost functional along a line between the computed point and the global minimizer, and then consider the value of the approximated cost gradient in this direction.

We then carry out a similar analysis in a plane, which allows us to conclude that the cost gradients are not being adequately approximated for this problem.

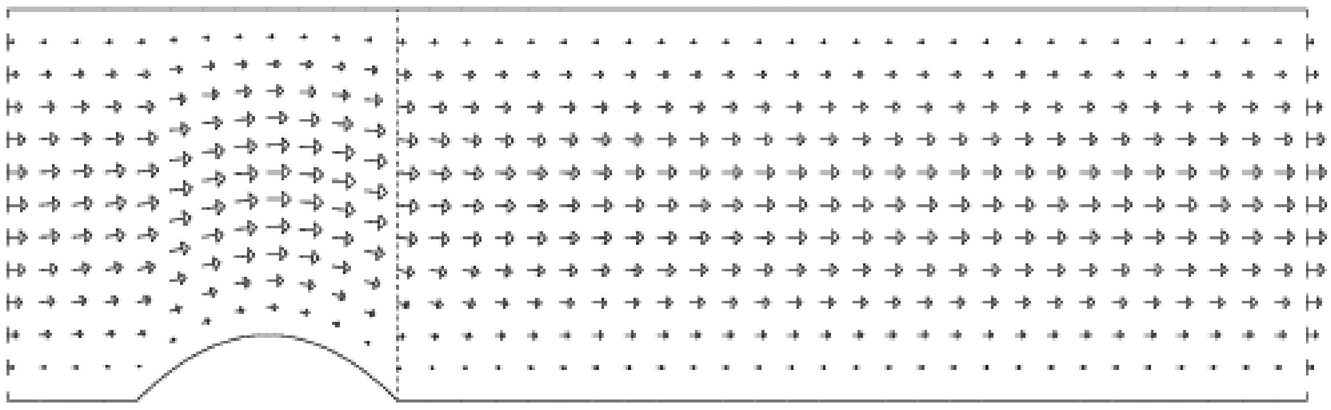

Figure 11.1: The target flow region, velocity field, and profile line.
The profile line has been moved to $x_s = 3$.

11.2 A Stalled Optimization

The experiments described in this chapter were all carried out on a problem with four free parameters: λ, representing the strength of the parabolic inflow, and α_1, α_2, and α_3, used to control the shape of a cubic spline that represented a bump extending from $x = 1$ to $x = 3$. The Reynolds number parameter Re was held fixed at a value of 1.

The target profile data was generated by computing a flow solution corresponding to an inflow parameter $\lambda^T = 0.5$ and shape parameters $\alpha^T = (0.375, 0.5, 0.375)$, and measuring the state variables along the profile line, which we have moved quite close to the bump, at $x_s = 3$, as shown in Figure 11.1. The optimization code was given a zero starting estimate for values of the parameters.

We define the cost functional in terms of the state variables as:

$$J_2^h(u^h, v^h, p^h, \lambda, \alpha) = \int_{x_s=3} (u^h(x_s, y) - u^T(x_s, y))^2 \, dy, \tag{11.1}$$

Table 11.1: Optimization results for $\mathcal{J}_2^h$.

TolOpt	Steps	λ	α_1	α_2	α_3	$\mathcal{J}_2^h$	$\|\|(\nabla\mathcal{J}_2)^h\|\|_\infty$
1.0E-09	33	0.505815	-0.0185	0.4352	-0.0448	0.390E-03	0.2E-01

and, in the usual way, define the constrained cost functional:

$$\begin{aligned} \mathcal{J}_2^h(\lambda,\alpha) &\equiv J_2^h(u^h(\lambda,\alpha), v^h(\lambda,\alpha), p^h(\lambda,\alpha), \lambda, \alpha) && (11.2)\\ &= \int_{x_s=3} (u^h(x_s,y,\lambda,\alpha) - u^T(x_s,y))^2 \, dy. && (11.3)\end{aligned}$$

The cost functional at the starting point was $\mathcal{J}_2^h(0,0,0,0) = 0.430$.

We approximate the cost gradient using discretized sensitivities. We present the result of the optimization in Table 11.1. The results presented are for an optimization tolerance of **TolOpt**=1.0E-9, but essentially identical results were achieved for tolerances of 1.0E-10, 1.0E-11 and 1.0E-12.

In this table, the value of the cost functional at the computed minimizer seems far too large. We happen to know that this value should be zero, although the optimization code has no way of knowing this. But more tellingly, the cost gradient is quite large. This is a serious problem, since the gradient must be zero at a minimizer, whether local or global, and the optimization code knows this. This means that it is very unlikely that the solution returned is even a local minimizer.

The picture of the computed flow in Figure 11.2 is disturbing, since we clearly have not achieved the secondary goal of reproducing the shape of the target bump.

The discrepancies between the known target and the solution returned by the optimization code are so serious that we must subject the solution to a series of tests to see if we are justified in discarding it, and to try to determine what went wrong.

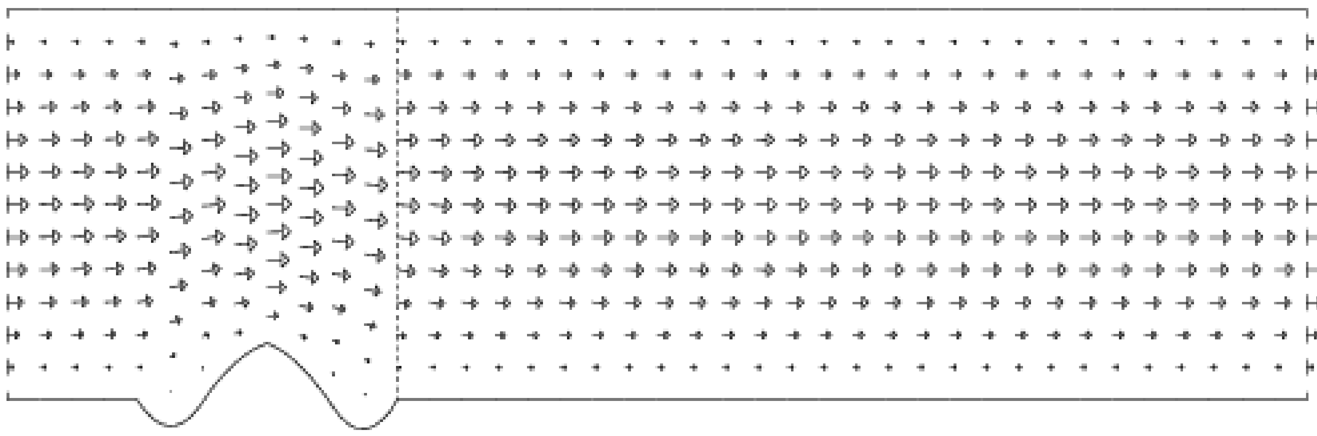

Figure 11.2: The flow produced with discretized sensitivities.

11.3 Checking the Results Along a Line

We are now going to consider the optimization results that we got using the discretized sensitivities in the computation of the cost gradient. We have already made three simple checks. We compared the cost functional value at the computed minimizer to the known minimum value of 0. The value of 0.390E-03 does not seem suitably small, particularly since we are carrying out a double precision calculation. The second check we made was to look at the gradient of the cost functional. The fact that the maximum entry has a size of 0.2E-01 is very disconcerting; at a true local or global minimizer, this value would be 0. Finally, we compared the computed bump values to the target values, and were not satisfied, although we have learned from the previous chapter that this is not always a reliable measure.

With our suspicions raised, it is time to try to gain a feeling for what is going on. While hardly conclusive, some graphic evidence may prove useful here. Let us simply draw a straight line in parameter space between the computed point and the target point, pick evenly spaced parameter values along this line, and evaluate the cost functional and the cost

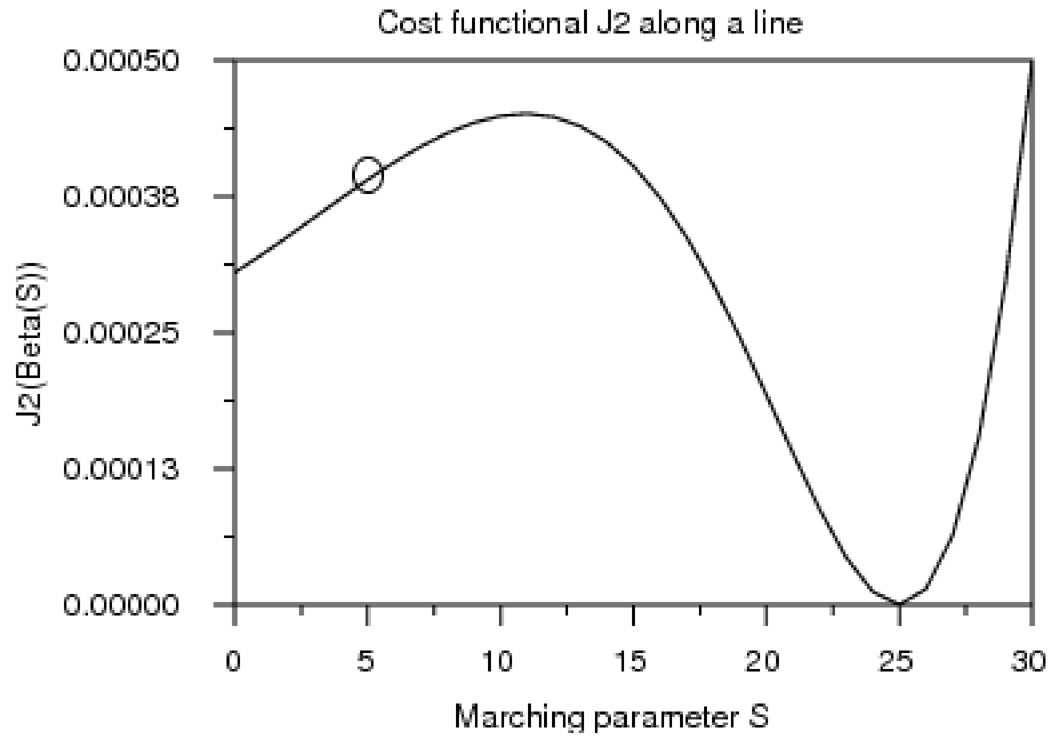

Figure 11.3: Values of $\mathcal{J}_2^h(\beta(S))$ along a line.
The "optimal" point is marked, at S=5.
The optimization used discretized sensitivities.
The local maximum on this line occurs at S=11.
The point S=25 is the global minimum.

gradient, as approximated with discretized sensitivities.

The actual table of values appears in Table 11.2, but perhaps the simple plot in Figure 11.3 best illustrates the problem. We must be careful to recall that the parameter space is actually four-dimensional, and that we are looking at a very narrow slice through it. Nonetheless, it is clear that something is quite wrong. It is surprising that there is a "hump" between the computed point and the global minimizer. If the computed point actually lies in a "valley" then the optimization, by its design, could never rise up out of the valley in order to reach the global minimizer. But this does not explain why the optimization did not then proceed downward to the left, where there are plenty of points with a lower functional value. The

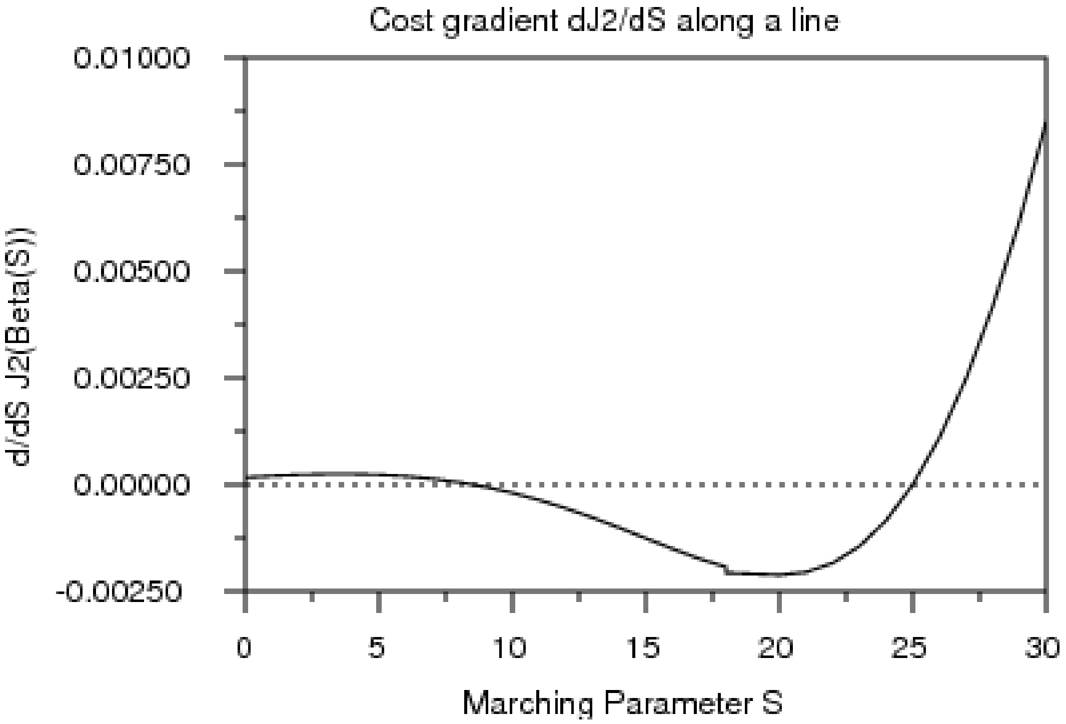

Figure 11.4: Values of $(\nabla \mathcal{J}_2)^h(\beta(S)) * dS$ along a line.
The optimization used discretized sensitivities.
Note that the computed slope changes sign between S=8 and S=9.

picture also shows us that the computed optimizer is not a local minimizer, nor does it seem to be an inflection point or local maximum. The behavior of the optimization code is still a mystery.

So let us now compute the gradient data that the optimization code had to work with. In Figure 11.4, we take the cost functional gradient, as approximated using the discretized sensitivities, and project it along the same line shown in Figure 11.3. Now some serious discrepancies are apparent. We see at local maximum near S=11, and a global minimum at S=25. Therefore, the correct slope must be positive up to S=11, and negative from there to S=25. But the computed slope is only positive up to S=8. Thus, there is an interval over which the cost functional is *increasing*, but the approximated directional derivative is *negative*. It is precisely this kind of discrepancy that will cause the optimization code to fail.

11.4 Checking the Results in a Plane

Our analysis of the optimization failure along a line has made it clear that something is wrong. In particular, it seems that the approximate gradients are inaccurate. But we'd like to get more evidence of the problem, to try to understand the context in which it occurs, how often it happens, and how severe the errors are.

To do so, we consider looking at the problem in a plane. That is, we start with the global minimizer and the computed minimizer, and specify some auxiliary point, and consider the plane that contains these three points. We then pick points on a regularly spaced grid over the plane, and evaluate the cost functional there. Finally, we make a contour map of the resulting data. The results are shown in Figure 11.5.

If we imagine a line drawn on this plot that goes from the computed minimizer to the global minimizer, then we can reproduce the one-dimensional plot in Figure 11.3. We can see that

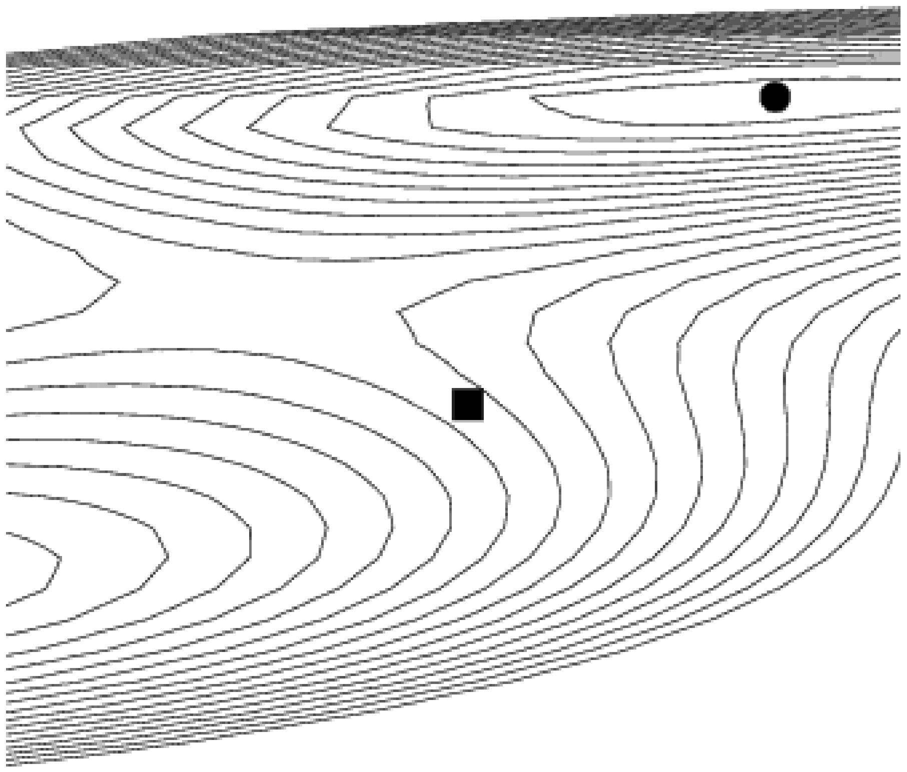

Figure 11.5: A contour map of the cost functional $\mathcal{J}_2^h$ in a plane.
The global minimizer is at the black dot.
The computed minimizer is at the black square.

the "hump" between the two points is, in part at least, avoidable, simply by moving to the side slightly. From this graph, it seems possible that this could be done is such a way that we did not have to increase the cost functional value.

It is quite interesting to note what seems to be a depression associated with a local minimum, in the lower left corner of the graph. Since we will have occasion to examine a similar phenomenon in the next problem that we study, we will overlook it for now.

From the contour plot alone, we cannot see what has gone wrong. But let us overlay this plot with the normalized gradient direction vectors, in Figure 11.6. We have taken the liberty of reversing the direction of the gradient field, so that it points *down*, in the direction of cost functional decrease.

We may compare this plot with Figure 11.7, generated by approximating the cost gradients with the finite difference sensitivities as in Equation (9.11). Consider the requirement that a gradient vector must cross a contour line at a right angle. The discretized sensitivity gradients fail this test, while the finite difference sensitivity gradients behave correctly. Note, moreover, that the discretized sensitivity gradients seem to be completely "unaware" of the local minimizer that lies off the page, in the lower left hand direction.

Our one-dimensional plot reported the same fact: for discretized sensitivities, the computed cost gradients do not reliably estimate the cost gradient. However, the two-dimensional plot gives a much better feel for the kind of details that can be missed, and for the magnitude of the error.

Accurate computation of the magnitude of the gradient is not as important as getting the *direction* correct. In Figure 11.8, we have magnified the portion of the region around the computed minimizer, and shown the gradient direction fields derived from both the discretized sensitivities and the finite difference sensitivities. We take the finite difference sensitivity

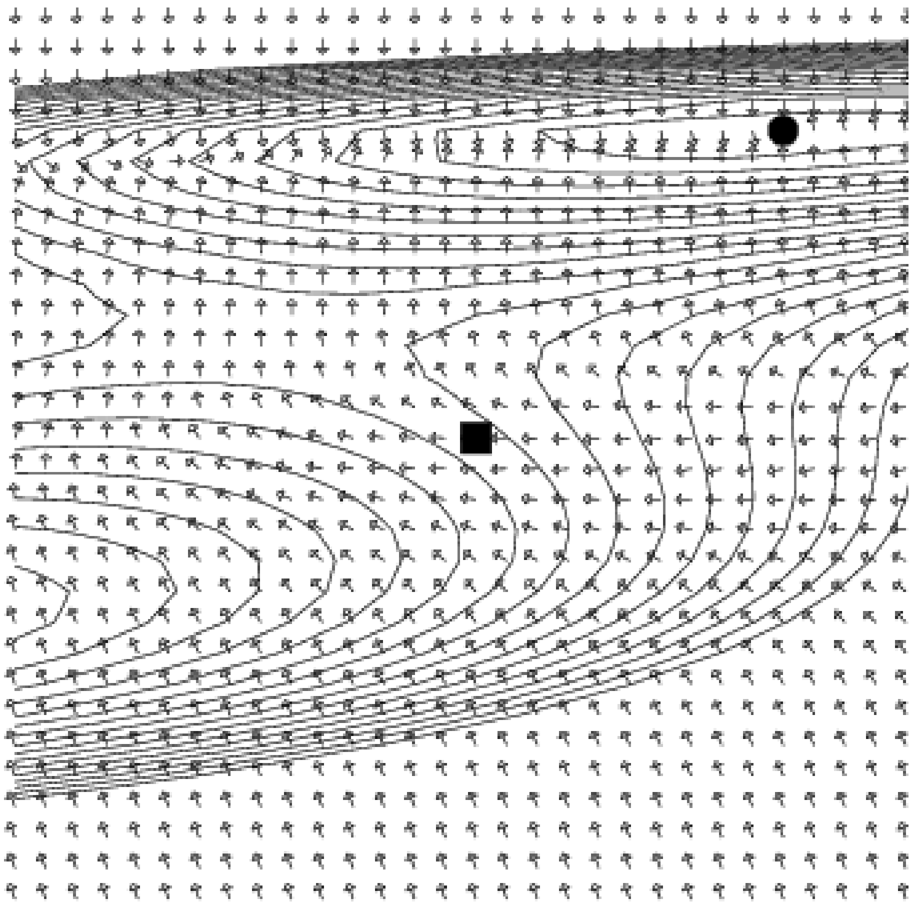

Figure 11.6: $\mathcal{J}_2^h$ gradient directions by discretized sensitivities.
The global minimizer is at the black dot.
The computed minimizer is at the black square.

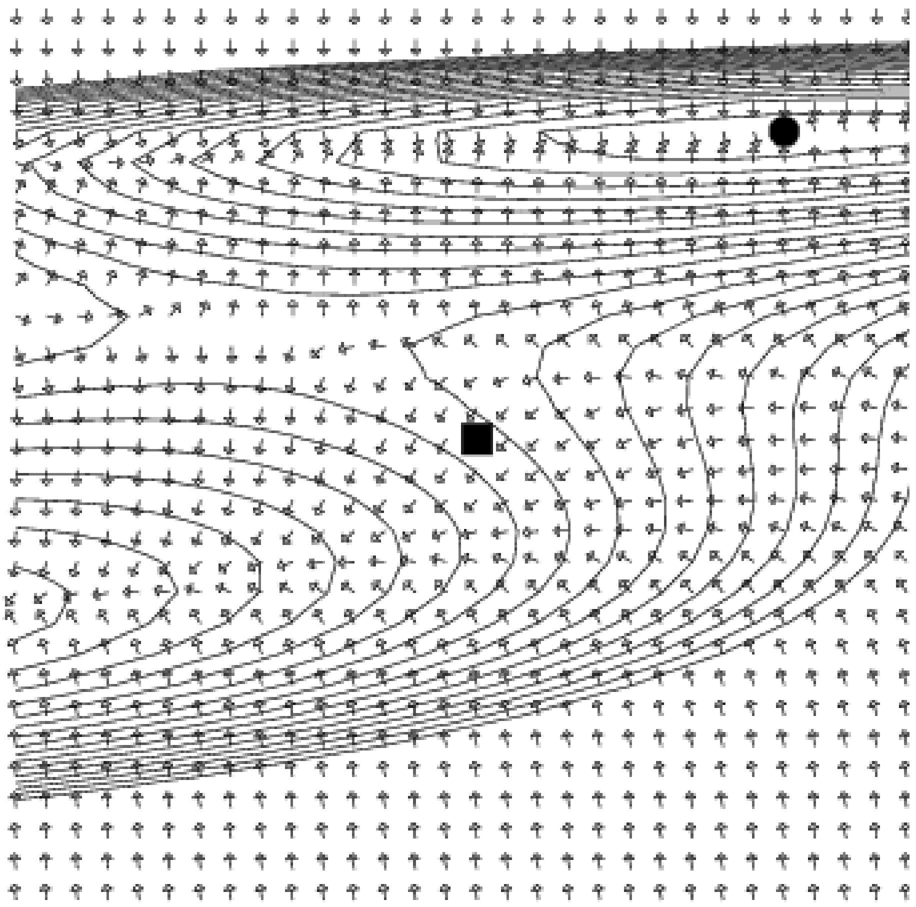

Figure 11.7: $\mathcal{J}_2^h$ gradient directions by finite difference sensitivities.
The global minimizer is at the black dot.
The computed minimizer is at the black square.

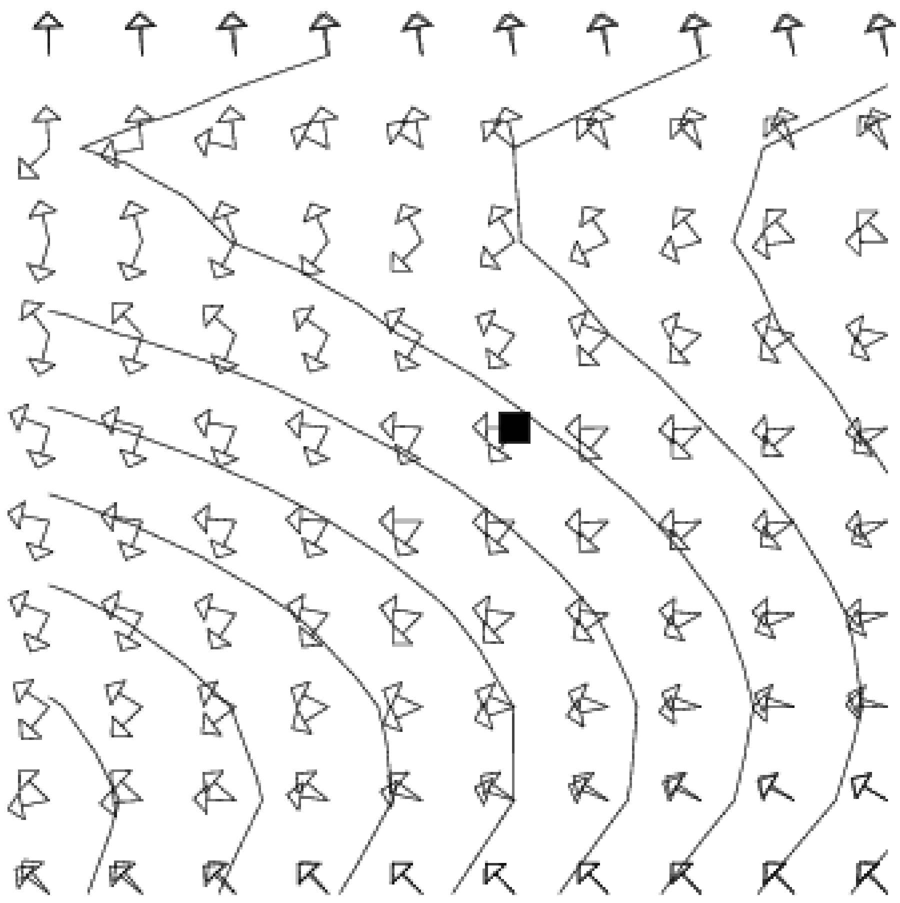

Figure 11.8: Comparison of $\mathcal{J}_2^h$ gradient directions.
Finite difference and discretized sensitivity approximations are shown.
The computed minimizer is at the black square.

gradients to be correct, using the contour lines as guides. Then it is clear that in this small region, there is an abrupt growth of error in gradient direction as we move from top to bottom. Within a few sample points of the computed minimizer, we see points at which the gradient direction error is greater than 90 degrees.

Of course, the evidence from this plot is not conclusive. We are only showing a two-dimensional slice of the parameter space. If we compute the maximum angle between the two four-dimensional gradient estimates, the largest value we compute over the set of points we are displaying is roughly 17 degrees. To actually understand how the accuracy of the gradient direction affects performance, we would have to look at the details of the particular optimization algorithm we are using.

But since we have seen that, at least for this “slice” of the problem, the cost gradients computed via finite difference sensitivities do such a good job of tracking the changes in the cost functional, it is natural that we should use them to repeat the optimization of the same problem we have analyzed here. We make this study in the next chapter.

Table 11.2: Values of $\mathcal{J}_2^h$ along a line in parameter space.
Point 6 is the optimizer returned when using discretized sensitivities.
Point 25 is the target parameters.
Note that a local maximum occurs between S=11 and S=12.

Point	λ	α_1	α_2	α_3	$\mathcal{J}_2^h$	$(\nabla \mathcal{J}_2)^h \cdot dS$
					$\times 1,000$	$\times 1,000$
0	0.507	-0.117	0.419	-0.149	3.05	0.16
1	0.507	-0.097	0.422	-0.128	3.21	0.20
2	0.506	-0.077	0.425	-0.107	3.38	0.23
3	0.506	-0.057	0.428	-0.086	3.55	0.24
4	0.506	-0.038	0.432	-0.065	3.73	0.24
5	0.505	-0.018	0.435	-0.044	3.90	0.23
6	0.505	0.001	0.438	-0.023	4.06	0.19
7	0.505	0.020	0.441	-0.002	4.21	0.13
8	0.504	0.040	0.444	0.018	4.33	0.05
9	0.504	0.060	0.448	0.039	4.42	-0.05
10	0.504	0.079	0.451	0.060	4.49	-0.19
11	0.504	0.099	0.454	0.081	4.51	-0.35
12	0.503	0.119	0.457	0.102	4.48	-0.55
13	0.503	0.138	0.461	0.123	4.39	-0.77
14	0.503	0.158	0.464	0.144	4.24	-1.00
15	0.502	0.178	0.467	0.165	4.03	-1.25
16	0.502	0.197	0.470	0.186	3.73	-1.50
17	0.502	0.217	0.474	0.207	3.37	-1.73
18	0.502	0.237	0.477	0.228	2.94	-1.93
19	0.501	0.256	0.480	0.249	2.45	-2.06
20	0.501	0.276	0.483	0.270	1.92	-2.11
21	0.501	0.296	0.487	0.291	1.39	-2.05
22	0.500	0.316	0.490	0.312	0.87	-1.83
23	0.500	0.335	0.493	0.333	0.43	-1.44
24	0.500	0.355	0.496	0.354	0.12	-0.84
25	0.500	0.375	0.500	0.375	0.00	0.00
26	0.499	0.394	0.503	0.396	0.14	1.10
27	0.499	0.414	0.506	0.417	0.63	2.49
28	0.499	0.434	0.509	0.438	1.55	4.18
29	0.498	0.453	0.513	0.459	2.97	6.19
30	0.498	0.473	0.516	0.480	4.99	8.51

Chapter 12

A LOCAL MINIMIZER

12.1 Introduction

In this chapter, we repeat the computations of Chapter 11, but now, in the computation of the cost gradients, the sensitivities are approximated by finite difference quotients rather than with discretized sensitivities.

We expect this approximation to be better, and indeed, the optimization code now converges properly to a minimizer. However, the minimizer is not the global minimizer corresponding to the target data, but is instead a local minimizer. We exhibit graphical evidence of a "barrier" between the this local minimizer and the global minimizer, which allows the local minimizer to trap the optimization code.

We then consider whether this local minimizer is a *spurious solution*, that is, a mathematical solution for the discrete problem which does not correspond to any physical solution. We carry out this investigation by refining the finite element mesh and examining the behavior of the refined finite element solution.

Table 12.1: Results for $\mathcal{J}_2^h$, using finite difference gradients.

TolOpt	Steps	λ	α_1	α_2	α_3	$\mathcal{J}_2^h$	$\|\|\nabla\mathcal{J}_2^h\|\|_\infty$
1.0E-09	44	0.507739	0.1407	0.5394	0.0599	0.311E-06	0.2E-09

12.2 Optimization Using Finite Difference Gradients

The experiments described in this chapter were all carried out on the same four-parameter problem used in Chapter 11. The algorithm was the same, except for one difference. Where previously we had estimated the sensitivities $(u^h)_\beta$ with the discretized sensitivities, $(u_\beta)^h$, we now computed direct finite difference quotient approximations $\dfrac{\Delta u^h}{\Delta \beta}$.

This is an expensive process, of course. Before, each step of the optimization iteration required one flow solve. Now, each step will require an additional flow solve for every parameter. In general, we would dearly like to avoid such wasteful computation by using discretized sensitivities. But for now, we need to do the extra work in order to understand whether the discretized sensitivities failed because of their limited approximation power.

The sensitivities, as estimated by finite differences, were used in Equation (9.11) to produce an approximation to the cost gradients $\dfrac{\partial \mathcal{J}_2^h}{\partial \beta_i}$. It was hoped that $\dfrac{\Delta u^h}{\Delta \beta}$ would give a better estimate of $(u^h)_\beta$, and that we would therefore arrive at a reliable approximation of the cost gradients.

As before, the optimization code was given a zero starting estimate for the parameters, and the cost functional at this starting point was $\mathcal{J}_2^h(0,0,0,0) = 0.430$.

We present the results of these computations in Table 12.1. The results presented are for an optimization tolerance of **TolOpt**=1.0E-9, but essentially identical results were achieved for tolerances of 1.0E-10, 1.0E-11 and 1.0E-12.

The first thing we check in these results is the parameter values, and we see that, as with

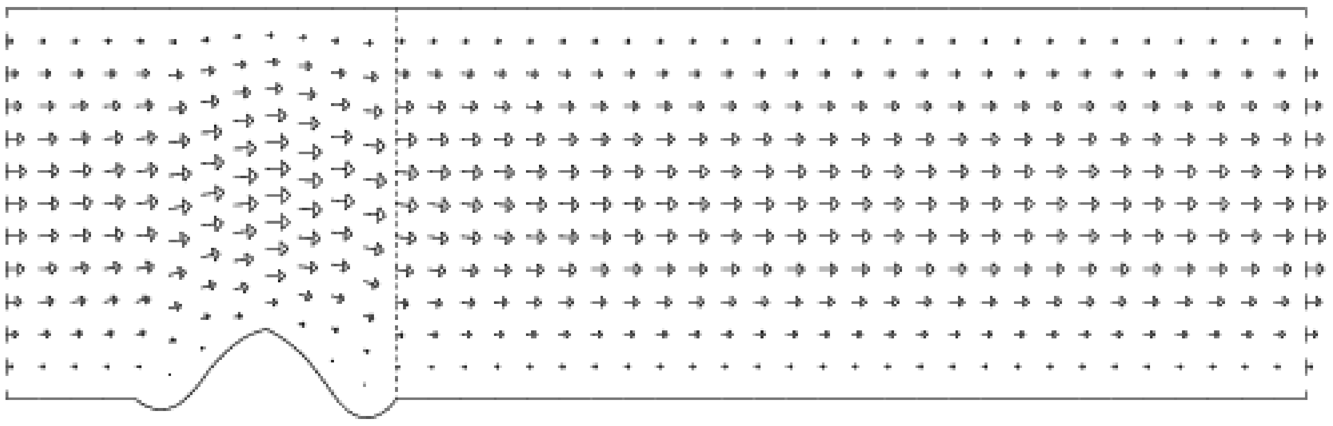

Figure 12.1: The local optimizing flow.
Gradients computed with finite difference sensitivities.
The mesh parameter h is 0.25.
This flow may be compared with the target flow, in Figure 11.1.

the discretized sensitivity calculation, we have been unable to reproduce the original target parameters.

We then turn to the cost functional information, and here we see that the optimization has probably been successful, at least in terms of returning a local minimizer. The cost functional has dropped by 6 orders of magnitude, from 0.430 to $0.311E-06$, and the gradient has dropped extremely close to zero, signaling that very little “local” improvement can be made to this solution point.

Thus, it seems we may have stumbled upon an unexpected solution to the problem. The parameters don’t seem far from the target values. We see in Figure 12.1 that the computed bump is not too far from the target bump, although it still has the “gutters” we saw in the previous chapter.

But before we accept this solution, we would like to repeat the tests of the previous chapter, and convince ourselves that what we are seeing this time is a real, acceptable, local minimizer for the optimization problem, and that this minimizer tells us something about the continuous physical problem whose behavior we are approximating.

12.3 Checking for Local Minimization

While it was natural to assume that the optimization code had halted at a local minimizer, there were some simple checks that could be made to verify this. For instance, it was possible that the optimization code had halted at the point because the functional was so flat that there was no discernible difference between functional values at that point and the global minimizer, or any points in a "plain" nearby. But this possibility was quickly dispelled by computing the cost functional at a series of intermediate points on the line between the optimization code's solution and the global minimizer. The graph in Figure 12.2 shows how the value of the cost functional rises from $\mathcal{J}_2^h = 0.3E-06$ at the optimization code's solution to an intermediate value of $\mathcal{J}_2^h = 0.7E-04$ before falling to $\mathcal{J}_2^h = 0$ at the global minimizer.

The graph strongly suggests that a local minimizer has been found. However, this is not guaranteed by any means. We have only sampled the functional along a direct line, whereas the space of parameters has four dimensions, meaning there could yet be a curve connecting the two points along which the functional is strictly decreasing. We would not expect the optimization software to miss an obvious descent direction, but we will feel more confident that we have found a local minimizer if we sample the functional in a plane and still see a barrier of higher functional values blocking access from the optimization code's solution to the global minimizer. Figure 12.3 displays such a plot, and the barrier is clearly visible. The solution returned by the optimization code lies in the "well" formed in the lower left hand corner, and the global minimizer is in the well in the upper right. Once the optimization code falls into the well in the lower left, it will not be able to rise out. Instead, its goal should be to seek the local minimizer in the well, which it does.

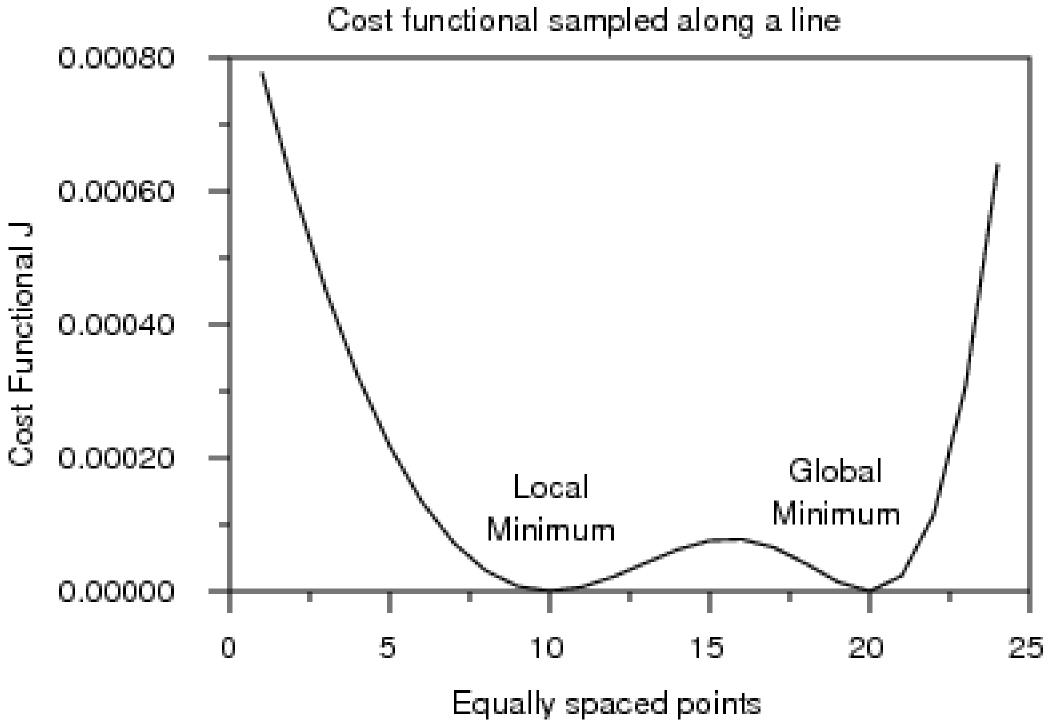

Figure 12.2: Values of $\mathcal{J}_2^h$ on a line between the local and global minimizers.

This graph, in which the computed minimizer correctly lies in the bottom of a local depression, should be compared with the unsatisfactory situation portrayed in Figure 11.5, where the optimization code stopped in confusion on a ridge, moving neither to the local nor global minimizers.

Finally, to convince ourselves that the cost gradient data is compatible with the cost functional, we display the (negative) normalized gradient vectors over the contour plot, in Figure 12.4. The plot makes it evident that the gradients computed using finite difference sensitivities correspond properly to the contour lines of $\mathcal{J}_2^h$, crossing them perpendicularly, and pointing inward toward the local and global minimizers.

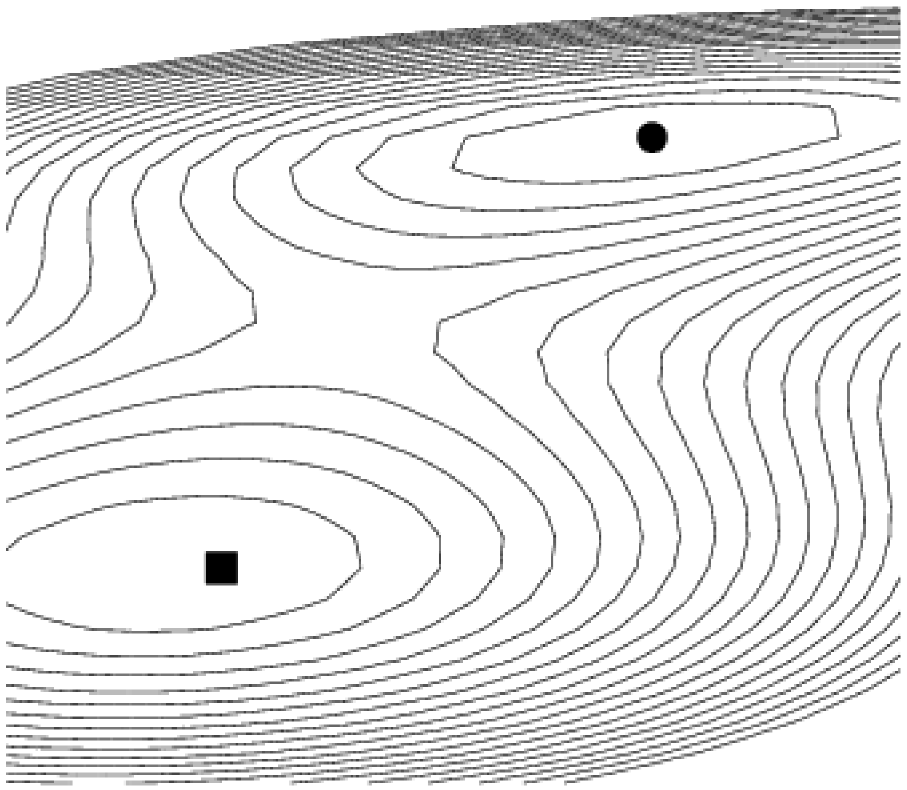

Figure 12.3: Contours of $\mathcal{J}_2^h$ on a plane containing the two minimizers.
The global minimizer is the black dot.
The local minimizer is the black square.

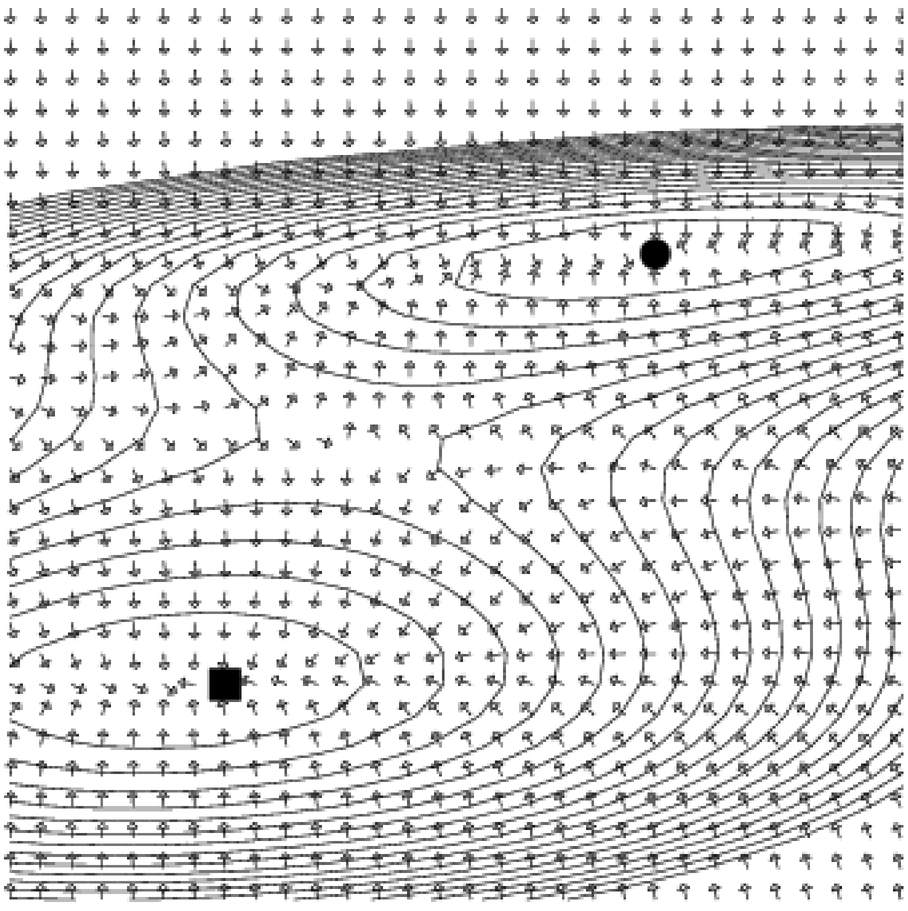

Figure 12.4: Contours and normalized gradients of $\mathcal{J}_2^h$.
Gradients computed using finite difference sensitivities.
The global minimizer is the black dot.
The local minimizer is the black square.

12.4 The Possibility of Spurious Solutions

Solving the discrete problem has given us two solutions to the minimization problem: the global minimizer that we had built into the problem, and the unexpected local minimizer. We have no doubt that the global minimizer found for the discrete problem corresponds precisely to the global minimizer that would be found if we could optimize the continuous problem. But it is not certain whether the local minimizer really has physical meaning, or is merely a *spurious solution*, that is, a mathematical artifact, a solution to the discrete problem that does not correspond to any physical solution.

It can be quite difficult to tell whether a given solution to a discretized problem has a physical meaning. For instance, we can convince ourselves that the "gutters" in the local minimizer's bump shape might perform the function of separating off regions of recirculating flow, so that some streamline actually comes close to reproducing the target bump shape.

On the other hand, we have grounds for disbelieving this solution. The guttering causes a substantial distortion of the relatively coarse finite elements, which makes the computational results in those elements less reliable.

One way to investigate this question is to examine the same problem on a sequence of finer meshes. If there actually is a true, physically meaningful, locally optimizing solution to the continuous problem corresponding to the discrete local minimizer we found, then we expect that each time we refine the mesh, we will again find a locally minimizing discretized solution, and that the sequence of these solutions will converge to the physical solution. Without actually knowing whether there is a true, limiting solution, we are testing the solutions by performing a rough sort of *Cauchy convergence test*, demanding that the successive iterates begin to cluster together.

The original solution was computed with mesh parameter $h = 0.25$, corresponding to 41

Table 12.2: Investigating the local minimizer with finer meshes.

h	λ	α_1	α_2	α_3	$\mathcal{J}_2^h(\lambda, \alpha)$
0.250	0.507739	0.14079	0.539404	0.059978	0.311E-06
0.166	0.508730	-1.46097	0.359595	-0.084067	0.104E-04
0.125	0.511107	-1.91279	0.298398	-0.290631	0.126E-04

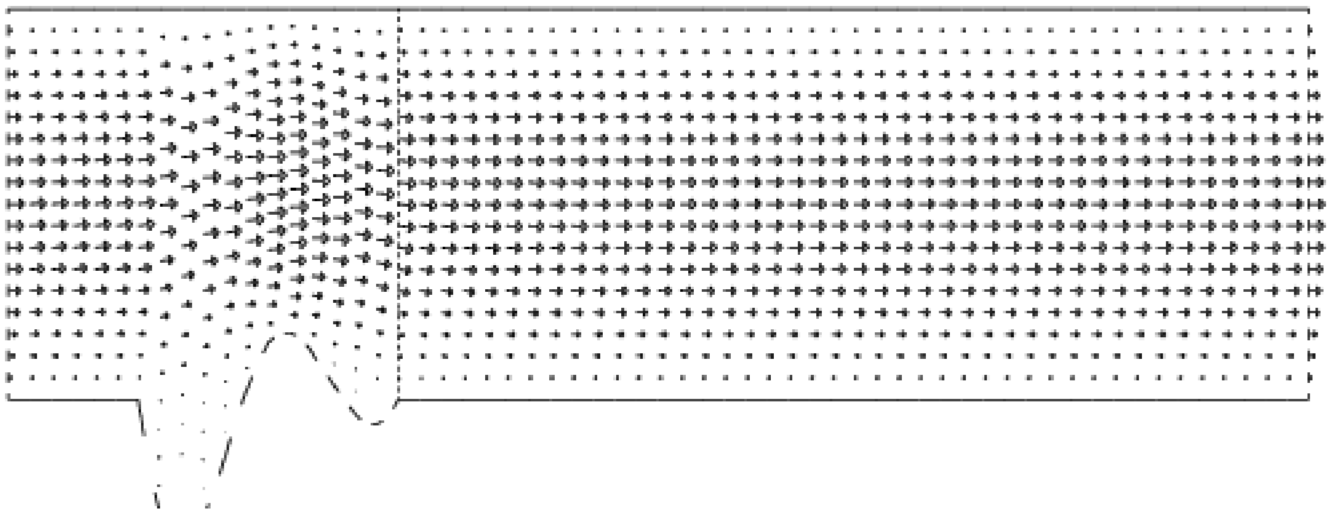

Figure 12.5: The flow for $h = 0.166$.

nodes along the x direction and 13 along y. To investigate the solution we found on this mesh, we set up two finer meshes, starting these optimizations at the local minimizer $\lambda = 0.507$, $\alpha = (0.140, 0.539, 0.059)$, and allowing up to 65 optimization steps. The numerical results are displayed in Table 12.2. In each case, the optimization code concluded that it had converged to an acceptable minimizer, but in both cases this minimizer had much deeper "gutters" than the original solution.

The solutions for $h = 0.166$ and $h = 0.125$ are shown in Figures 12.5 and 12.6. These flows may be compared with the flow solution for $h = 0.125$ in Figure 12.1.

The results from these runs on finer meshes suggest that the local minimizer is a spurious solution. Instead of staying put, the gutters have gotten much deeper. The fact that the

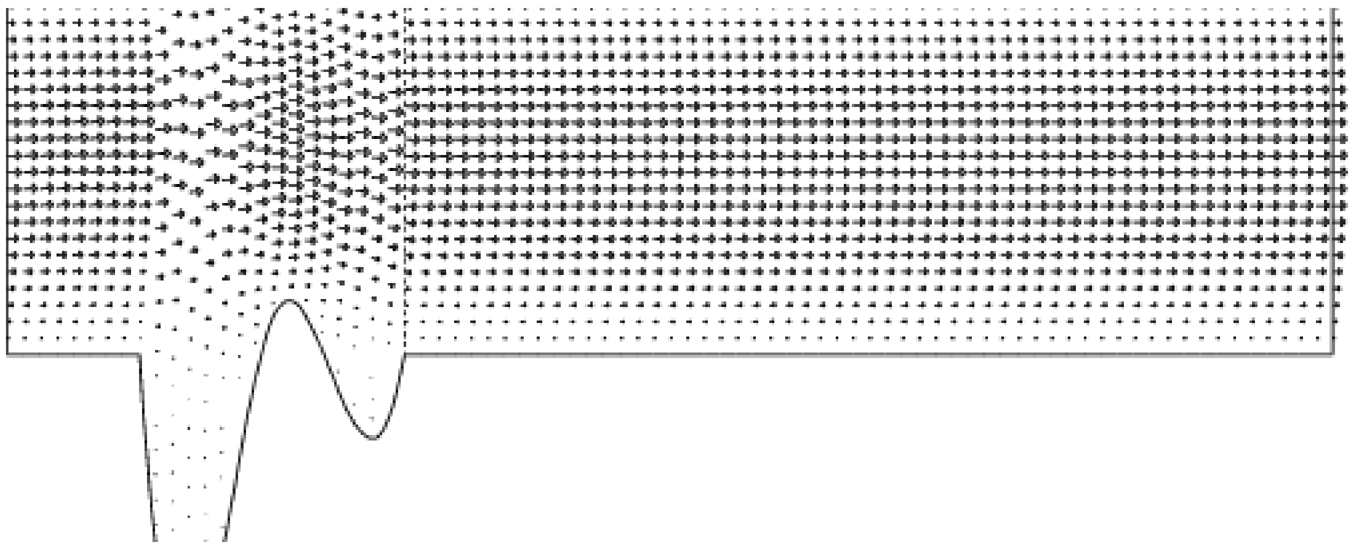

Figure 12.6: The flow for $h = 0.125$.

gutters are so deep and narrow also means that all the elements above the gutters are severely distorted, so that we cannot place much confidence in the computed flow results either. Since this local minimizer doesn't seem to represent an interesting physical solution, we are no longer interested in it, except as a trap which we wish to avoid. We now consider how to modify the approach so that the optimization can reach the global minimizer, avoiding such meaningless solutions as we have just encountered.

Chapter 13

OPTIMIZATION WITH A PENALTY FUNCTIONAL

13.1 Introduction

In this chapter, we try to avoid the local minimizer we found in Chapter 12 by adding a "penalty functional" to the cost functional. As described in Section 9.11, such a penalty functional can regularize a problem, that is, disqualify certain solutions with undesirable properties; in particular, we prefer solutions with a monotonic bump derivative, and we will use the penalty functional to try to select such solutions over those with "wiggles" or "gutters".

It is hoped that this regularization will fill in the valley of attraction for the local minimizer seen in the previous chapter, and allow the optimization code to reach the global minimizer. We will see that regularization will allow us to come closer to the global minimizer, but a more elaborate procedure is needed to actually reach that point.

13.2 Regularizing the Cost Functional

We have already computed a number of solutions to the optimization problem, and found something objectionable with each. The biggest drawback, of course, was that none of these solutions were the global minimizer.

In a real problem, when the optimization code returns a purported minimizer, we won't know if there is a better, global minimizer. Nonetheless, there are some criteria that can be used to declare that a solution is unacceptable, or undesirable.

For the problems that we have been examining, a wind tunnel engineer might reject every solution with negative bump parameters because the concavities cause turbulence, are too hard to machine, or are simply implausible. Solutions in which the bump almost entirely occludes the channel might also be rejected for practical reasons.

We leave ourselves open to such undesirable results since we haven't placed any constraints on the feasible space of parameters to be examined by the optimization code. We could explicitly add new requirements, in the form of equalities or inequalities that the parameters would have to satisfy. However, this would change the problem to one of *constrained optimization*, adding a layer of complication we prefer to avoid.

It is enough, for our purposes, if we can *discourage* the optimization code from making certain choices, rather than forbidding them. This calls for the use of a penalty function, which allows the optimization code to investigate any set of parameters, but which attempts to make some choices unattractive by artificially increasing the cost functional there.

One possibility for controlling the oscillations of the bump is to compute a penalty functional $\mathcal{J}_B^h(\alpha)$ based on the integral of the square of the derivative of the bump:

$$\mathcal{J}_B^h(\alpha) = \int_1^3 (\frac{\partial Bump(x,\alpha)}{\partial x})^2 \, dx. \tag{13.1}$$

Table 13.1: Optimization results on $\mathcal{J}_3^h$ for various values of ϵ.

ϵ	Steps	λ	α_1	α_2	α_3	$\mathcal{J}_B^h$	$\mathcal{J}_3^h$
1.0	22	0.512681	0.0026	0.0044	0.0032	0.4E-04	0.8E-02
1.0E-01	11	0.512231	0.0273	0.0467	0.0356	0.5E-02	0.8E-02
1.0E-02	19	0.505713	0.1740	0.3262	0.2706	0.2E+00	0.4E-02
1.0E-03	31	0.500561	0.2515	0.4677	0.3665	0.5E+00	0.5E-03
1.0E-04	40	0.500382	0.2849	0.5001	0.3636	0.5E+00	0.6E-04
1.0E-05	47	0.505597	0.1305	0.5442	0.1527	0.1E+01	0.1E-04
1.0E-06	47	0.507467	0.1387	0.5406	0.0712	0.6E+00	0.1E-05
1.0E-07	43	0.507710	0.1405	0.5395	0.0611	0.6E+00	0.4E-06
0.0	44	0.507739	0.1407	0.5394	0.0599	0.6E+00	0.3E-06

Such a penalty functional would be zero for a straight line connecting the beginning and ending of the bump region, low for a small parabola or a piecewise linear function, and high for a curve with wiggles or severe curvature. Our optimization code would then work with a new cost functional $\mathcal{J}_3^h$ defined by adding a "small" multiple ϵ of the bump oscillation integral to the original cost:

$$\begin{aligned}\mathcal{J}_3^h(\lambda,\alpha) &\equiv \mathcal{J}_2^h(\lambda,\alpha)+\epsilon\mathcal{J}_B^h(\lambda,\alpha) &(13.2)\\ &= \int_{x_s=3}(u^h(x_s,y)-u^T(x_s,y))^2\,dy+\epsilon\int_1^3(\frac{\partial Bump(x,\alpha)}{\partial x})^2\,dx. &(13.3)\end{aligned}$$

Here, we don't include ϵ as a parameter of $\mathcal{J}_3^h$, because we imagine it as being fixed to some suitable value during the computation.

The optimization code minimizes this new cost functional by attempting to decrease the flow discrepancy while not allowing the bump oscillation integral to get too large.

We ran a series of optimizations on the usual four parameter problem, with the target parameters at $\lambda^T=0.5$, $\alpha^T=(0.375,0.5,0.375)$. We considered a series of values of ϵ to try to get an idea of the effect of the penalty term on the computed optimizing solution. The results are summarized in Table 13.1.

In this table, it is interesting to see how, as ϵ decreases, the computed solution begins to

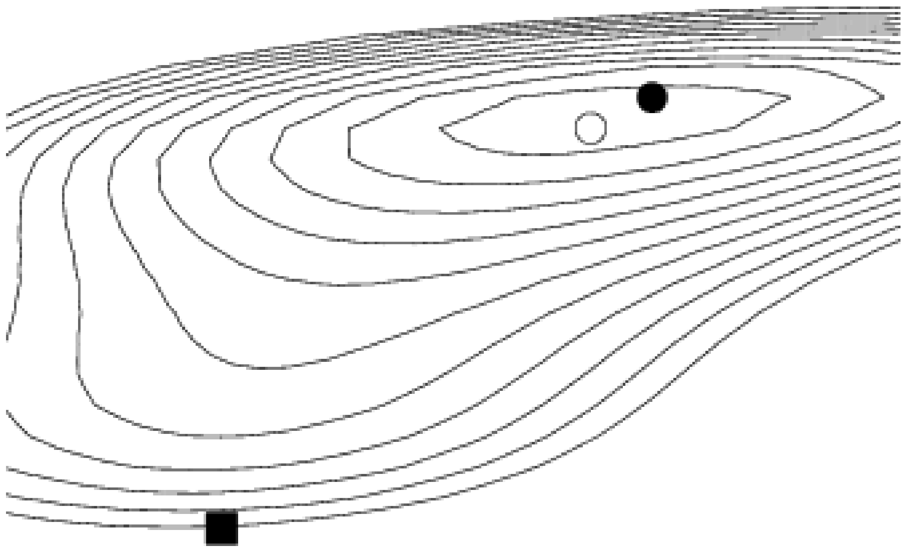

Figure 13.1: Contours for cost functional $\mathcal{J}_3^h$ with $\epsilon = 0.0002$.
The open circle marks the minimizer of $\mathcal{J}_3^h$.
The filled circle marks the global minimizer of $\mathcal{J}_2^h$.
The square marks the local minimizer of $\mathcal{J}_2^h$.

move towards the global minimizer, down to the value ϵ=0.0001. Then, the local minimizer becomes more attractive, and as ϵ decreases further, the computed solution begins to approach that point instead. Of course, if we actually set ϵ to zero, then that's precisely where we end up. From the table, we can conclude that the penalty functional can help, for *certain* values of the penalty parameter ϵ; the value mustn't be too large (or else we're not solving the flow problem we're interested in) but it mustn't be too small (or we'll end up approaching the undesirable local minimizer).

To get an idea of the smoothing effect of this change in the functional, consider Figure 13.1, which displays contour lines of the smoothed functional, for $\epsilon = 0.0002$, over the same two dimensional plane used in Figure 12.3. The addition of the bump term seems to have completely smoothed away the valley containing the local minimizer.

However, the minimizer of the new functional $\mathcal{J}_3^h$ is *not* the minimizer of the previous functional $\mathcal{J}_2^h$. This is to be expected; we could only guarantee that the minimizers were the same in the unlikely case that the global minimizer of $\mathcal{J}_2^h$ also happened to minimize the penalty function.

13.3 Reaching the Global Minimizer

Since we didn't reach the global minimizer, it might seem that our penalty functional approach has not solved our problem. While our real goal was to reach the global minimizer of $\mathcal{J}_2^h$, we have reached a minimizer of a related functional, which is merely *near* our desired value.

But it was exactly the task of getting near the global minimizer that we could not accomplish earlier. Just as we would get near, the local minimizer would attract the optimization code, and once we entered the local minimizer's "valley" there was no way to escape.

The penalized functional $\mathcal{J}_3^h$ gives us a way to restart such a computation from the local minimizer with the hope of exiting the valley. For values of ϵ that aren't too small, the valley disappears. Assuming that the global minimizer has a lower value of $\mathcal{J}_3^h$, (though not necessarily the *lowest*), it is reasonable to assume that a few steps of optimizing $\mathcal{J}_3^h$ will move us closer to the global minimizer. Then if we return to optimizing $\mathcal{J}_2^h$, we have a chance of reaching the desired solution.

As our first example of this approach, let us start an optimization of $\mathcal{J}_2^h$ at the usual starting

Table 13.2: Results at the end of each stage of the three-stage optimization.

Minimizing	Steps	λ	α_1	α_2	α_3	$\mathcal{J}_2^h$	$\mathcal{J}_3^h$
$\mathcal{J}_2^h$	44	0.507739	0.1407	0.5394	0.0599	0.311E-06	0.297E-03
$\mathcal{J}_3^h$	15	0.502012	0.3411	0.5380	0.3086	0.144E-04	0.145E-03
$\mathcal{J}_2^h$	26	0.500007	0.3743	0.5000	0.3748	0.498E-09	NA

point. When that optimization is completed by reaching the local minimizer, let us use that point as the starting point of a new optimization of $\mathcal{J}_3^h$, with $\epsilon = 0.0002$. We don't need to solve this problem well; we just want to get away from the local minimizer, so let's take 15 steps. Then we will use *that* set of parameters as the starting point for a third optimization of our unpenalized cost functional $\mathcal{J}_2^h$. Essentially, we're using the penalized function to " get over the mountain" between the local and global minimizers.

As Table 13.2 makes clear, our 15 steps of minimizing $\mathcal{J}_3^h$ actually *raise* the value of $\mathcal{J}_2^h$ as it moves away from the local minimizer. But this is exactly what must happen if we are to climb out of the local valley. Once we are on the other side, it is safe to return to minimization of $\mathcal{J}_2^h$, and we finally have the satisfactory result of reaching our global minimizer.

Even though we are optimizing $\mathcal{J}_3^h$, there is nothing that says we can't monitor the values of $\mathcal{J}_2^h$. If we do so, then we can take a sustained drop in the cost functional value as a sign that we have made it to the other side of the ridge. The actual sequence of cost functional values for the 15 iterates in our case is shown in Figure 13.2.

If we anticipate problems with a local minimizer, or we want to try to rule out certain behaviors with a penalty functional, then we could always begin by seeking a minimizer of the penalized functional $\mathcal{J}_3^h$. Then we restart the optimization from that point seeking the minimizer of $\mathcal{J}_2^h$. The results of carrying out this procedure on our current problem are shown in Table 13.3. Again, we used a value of $\epsilon = 0.0002$ to define $\mathcal{J}_3^h$. As can be seen, the initial optimization stops quite close to the global minimizer, making the second stage easy.

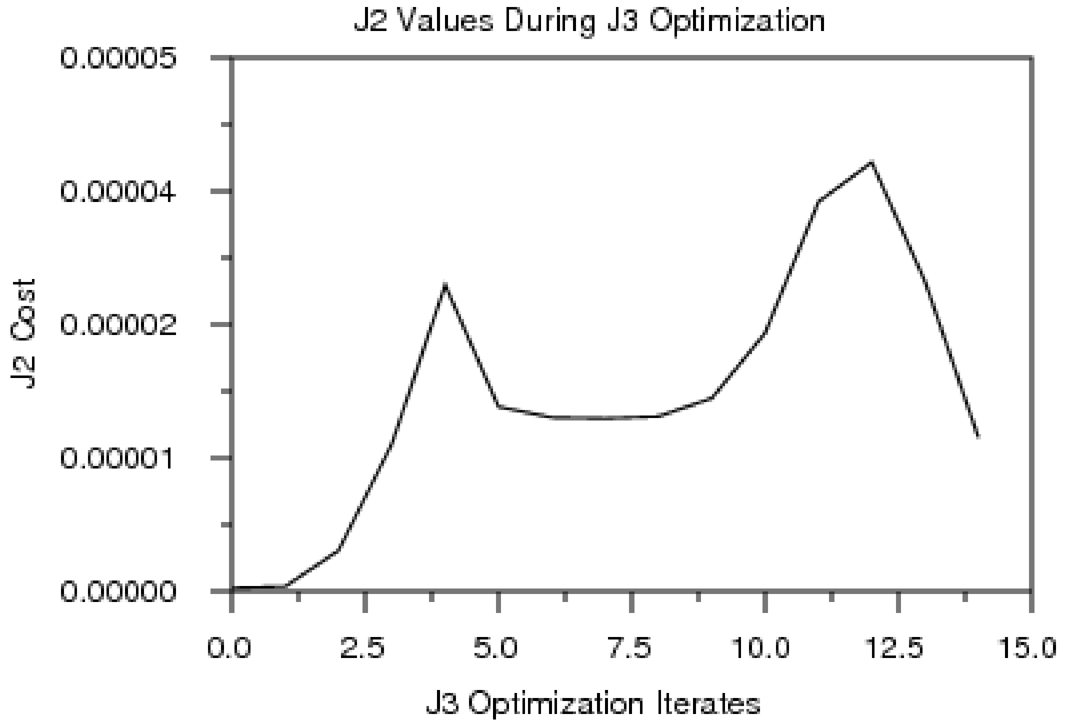

Figure 13.2: Values of $\mathcal{J}_2^h$ during the optimization of $\mathcal{J}_3^h$.

Table 13.3: Results at the end of each stage of the two-stage optimization.

Minimizing	Steps	λ	α_1	α_2	α_3	$\mathcal{J}_2^h$	$\mathcal{J}_3^h$
$\mathcal{J}_3^h$	45	0.500363	0.2764	0.4952	0.3649	0.194E-05	0.119E-03
$\mathcal{J}_2^h$	26	0.500015	0.3735	0.5001	0.3747	0.283E-09	NA

Thus we have seen that, in the presence of an attractive local minimizer, a penalty functional may be used to escape from the local region of attraction, or, if used in advance, to avoid it altogether. A number of *ad hoc* decisions must be made; the most important one being the choice of the form of the penalty functional. Typically, this choice is guided by seeking to eliminate objectionable or unphysical features of the local minimizer. Secondly, an appropriate weight ϵ must be chosen when the penalty functional is added to the cost. Finally, the user must decide when to switch between optimization of the penalized and unpenalized cost functionals.

Chapter 14

FLOWS AT HIGHER REYNOLDS NUMBERS

14.1 Introduction

Most of our work has been done at a Reynolds number $Re = 1$; real fluid flows encountered in aircraft engine design may have Reynolds numbers of 10,000, 100,000 or more. We discuss in this chapter some of the changes expected in the behavior of the flow and in the form of the sensitivities as Re increases. Because higher Reynolds number flows are inherently more difficult to compute, we discuss a continuation method approach for producing a suitable starting point for the Newton iteration.

14.2 "Local" Influences Become Global

Low Reynolds number flow could be characterized by honey moving through a tube, while an example of high Reynolds number flow would be air driven at high speed down a tunnel. Experience tells us that the high Reynolds number flow is inherently more unpredictable and unsteady. We are interested, though, in the behavior of the two kinds of flows when an obstacle is encountered. We have already seen that, for $Re = 1$, the velocity sensitivities are relatively large near the bump, but die out very quickly downstream. When we plotted the sensitivities, we saw a "whirlpool" of influence right around the bump, and little else. We interpreted this to mean that the bump caused almost purely a local disturbance.

It is natural to suppose, again from physical experience and intuition, that as the Reynolds number increases, the influence of the bump will extend to a greater proportion of the flow region behind the bump, while decreasing somewhat in front of the bump. In fact, at higher Reynolds numbers, the influence of *all* the parameters and boundary conditions begins to affect the solution throughout the region.

This is good news for our optimization efforts; it means that more of the flow will be affected by the bump, and we will have a greater choice on where we place our profile line, or, indeed, on what quantities we measure there.

On the other hand, it is going to make our individual computations of particular flow solutions much harder; as Re increases, the nonlinear terms begin to dominate and our problem, which we symbolize abstractly as $F^h(X) = 0$, becomes much harder to solve.

As Re increases, $F^h(X) = 0$ becomes not only a harder problem, but also a *bigger* problem, because the mesh parameter h must be decreased. Cutting h in half quadruples the number of unknowns, and our banded system matrix increases in size by roughly a factor of 8. Such growth factors cannot be long sustained, and severely limit the Reynolds numbers for which we can compute an approximate flow solution. But if the mesh is not sufficiently refined for a given Re, the Newton iteration may converge extremely slowly, or, more likely, will almost immediately diverge.

We now exhibit a set of plots of the α-velocity sensitivity field for Re=1, 10, 100, 500 and 1,000. These plots confirm the prediction that the influence of the bump begins to extend downstream; it is almost as though the "whirlpool" that showed up above the bump, for

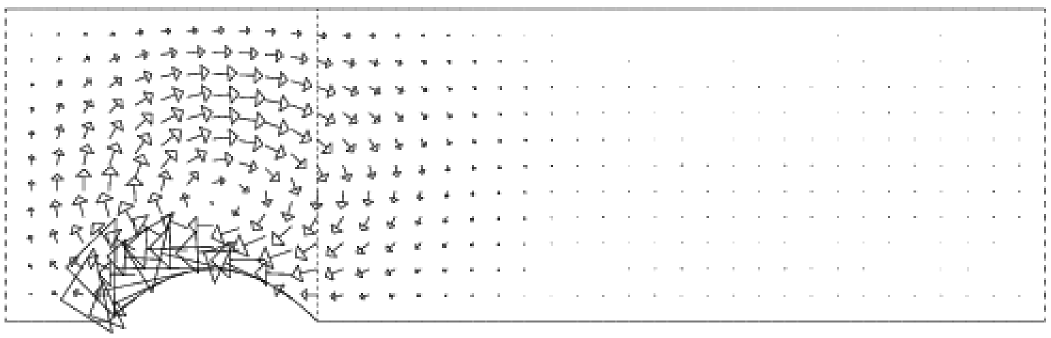

Figure 14.1: Velocity sensitivities for $Re = 1$.

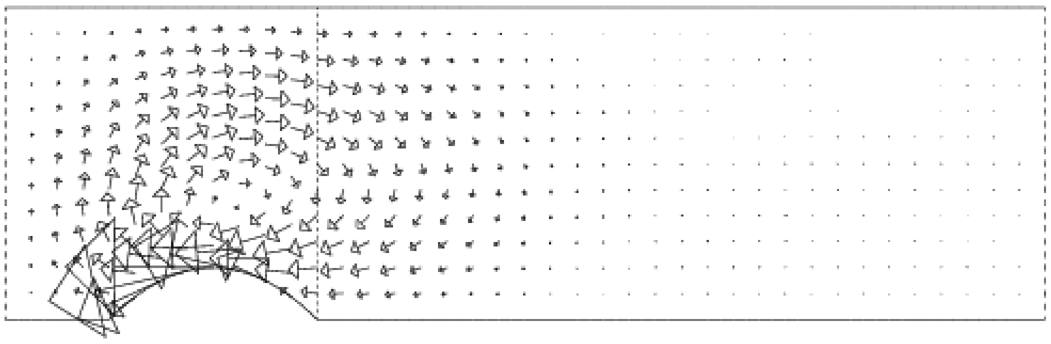

Figure 14.2: Velocity sensitivities for $Re = 10$.

$Re = 1$, is blown to the right by an incoming wind.

A comparison of the sensitivity plots for $Re = 1$ and 1000 suggests that, at $Re = 1$, the disruptions that the bump causes are strictly confined to the region above the bump, where more flow has to pass through a narrowed channel; once past the bump, the flow quickly restores itself to its original pattern. For $Re = 1000$, however, the flow deflected upward does not come back down. The large arrows in the wake of the bump indicate that the bump is causing the flow just behind it to die off.

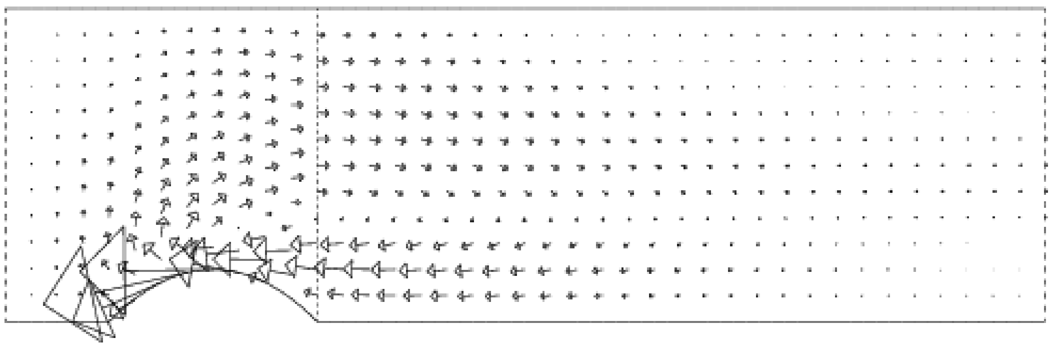
Figure 14.3: Velocity sensitivities for $Re = 100$.

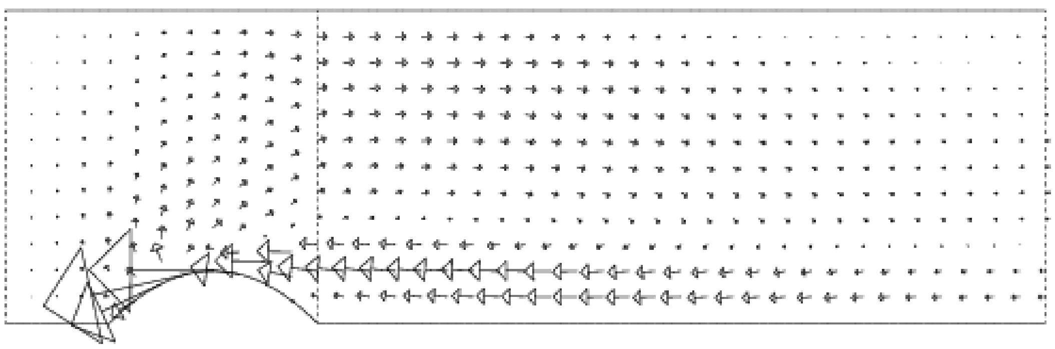
Figure 14.4: Velocity sensitivities for $Re = 500$.

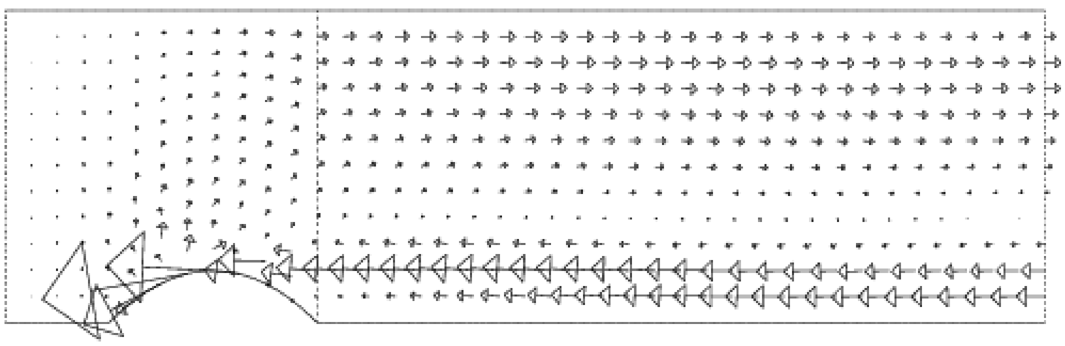
Figure 14.5: Velocity sensitivities for $Re = 1000$.

14.3 Getting a Starting Point

At low Reynolds numbers, we started our first Newton process with a zero guess, and thereafter simply used the previous solution for the next Newton process. At higher Reynolds numbers, the nonlinear terms are more significant. This means that Newton's method has to work harder, and requires a better starting point. It's easy to detect when the starting point isn't good enough, because the Newton iteration will begin to fail.

If a better starting point is needed, one option is to compute a solution using a simple form of *continuation* [23]. That is, to get the solution (u, v, p) for the parameters $(\lambda_0, \alpha_0, Re_0)$, we first compute the solution for $(\lambda_0, \alpha_0, 1)$. For example, if Re_0 is "close" to 1, and we have a flow solution computed at $Re = 1$, then we can compute a good initial estimate of the solution at Re_0 by using an Euler approximation:

$$u(\lambda_0, \alpha_0, Re_0) \approx u(\lambda_0, \alpha_0, 1) + \frac{\partial u}{\partial Re} * (Re_0 - 1). \tag{14.1}$$

Similar formulas are required to approximate the new values of v and p. Notice that, once again, we need the sensitivities of the solution variables, this time with respect to Re. Of course, we can estimate the sensitivities by finite differences, or try to economize by using discretized sensitivities.

If Re_0 is not close to 1, then the Euler approximation might not produce a good enough starting point for the Newton iteration. Then a possible rememdy would be to advance to Re_0 in a sequence of several steps. At each step, we use an Euler approximation based at the previous step as the starting point for our Newton process.

Once we have gotten a single solution (u, v, p) at our parameter values $(\lambda_0, \alpha_0, Re_0)$, we may choose to reuse that solution as the starting point for the next Newton process. However, at high Reynolds numbers, the solution is much more sensitive to the parameter values, and we may find this approximation unsatisfactory. In that case, we must return to the continuation method. If the Newton process fails during a continuation step, we always have the option of reducing the stepsize and trying again.

We have used this approach successfully to compute target solutions at Reynolds numbers of 100 and 1,000. However, during the subsequent optimization steps, we have encountered numerous problems with Newton divergence. In some cases, the bump parameter values become strongly negative, or oscillatory; in other cases there is no clear reason for the failure. Frequently, a convergence could be corrected by refining the mesh. However, because of computer memory limitations, it was not possible for us to use a mesh parameter h finer than 0.0625, which meant we frequently were unable to solve problems at $Re = 1000$.

Chapter 15

APPROXIMATING NONFEASIBLE TARGETS

15.1 Introduction

The fundamental information defining our cost functional is the profile data, that is, the values $u^T(x, y)$ (and possibly $v^T(x, y)$ and $p^T(x, y)$) specified along the profile line $x = x_s$. Usually, this data is generated by specifying some set of parameters β^T, solving for the corresponding flow $(u^T(\beta), v^T(\beta), p^T(\beta))$, and saving the flow values along the profile line.

In such a case, we speak of (u^T, v^T, p^T) as a *feasible target.* This terminology is meant to imply that the profile data was generated from a discrete fluid flow, that this fluid flow was generated by solving a parameterized flow problem, and that the original set of parameters is part of the feasible set of parameters to be examined by the optimization code. In other words, if the target is feasible, the generating parameters are buried somewhere in the feasible set, if only the optimizer is smart enough to find them.

Hence if we generate our problems in this way, we know beforehand that the minimum value of the discrepancy cost functional will be zero, and that this minimum value certainly occurs for the flow solution corresponding to the parameter values β^T, although it may occur for other parameter values as well. For that matter, we could also have numerous local minimizers with higher values of the cost.

However, there is no requirement that we set up our cost functional data in this way. In a real world application, it is likely that we would not know the form of any global or local minimizer beforehand. It is also likely that we would prescribe what might be termed an *unfeasible target*; that is, profile data for which there is no corresponding set of parameters that would produce a flow with that profile.

We would like to investigate some simple problems involving unfeasible targets. We will consider some problems where the target data is generated from a flow that depends on parameters, but where the spaces spanned by the parameters are different than those used for the optimization. We will also consider some problems where the target data is simply produced, *data ex machina*, with presumably no corresponding flow solution.

In these cases, we will still expect the optimization code to produce a locally minimizing solution, but we have no idea beforehand what the minimal value will be, nor what the form of the solution will be. In cases where our target data is chosen too arbitrarily, we may find that the optimization results are counter-intuitive.

15.2 Example 1: Differing One-Parameter Bump Shapes

For our first example, let us suppose that the target flow is generated in a flow region whose bump is a piecewise quadratic, whereas the feasible space will only consider bumps which are piecewise linear. To keep things simple, we will only use a single bump parameter, $\alpha = 0.5$, and we will fix the inflow parameter α at 0.5, and the Reynolds parameter at 1. When we give this problem to our program, the optimizer converges to the solution $\alpha = 0.6498$ in 14 steps, with a cost functional value of $\mathcal{J}(\alpha) = 0.62E-4$. Of course, we didn't expect the two

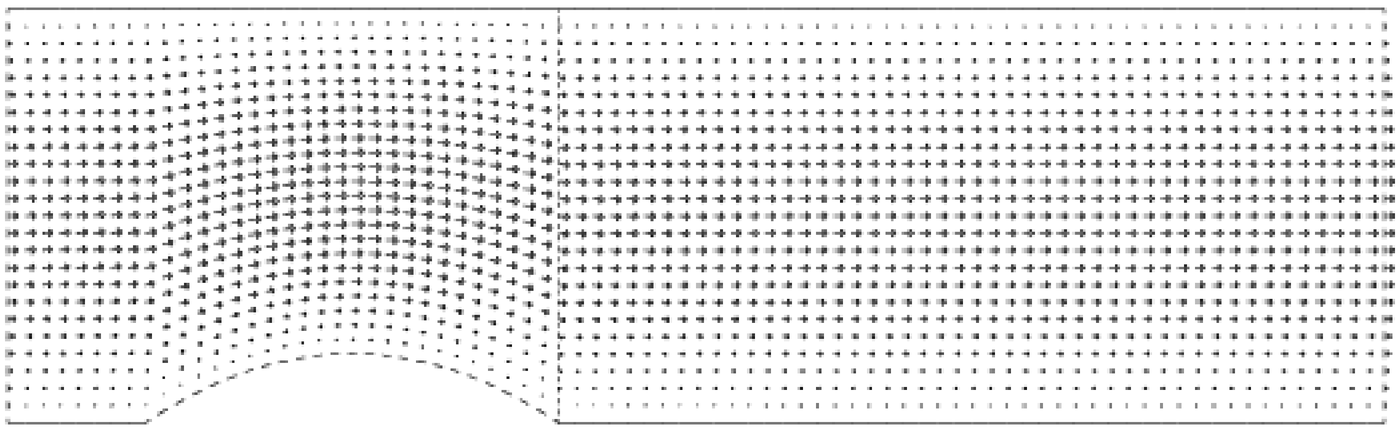

Figure 15.1: Example 1: The target flow.
A piecewise quadratic bump is used.

Table 15.1: Example 1: Optimization results.

Steps	α	$\mathcal{J}_2^h$	$\|\|\nabla\mathcal{J}_2^h\|\|_\infty$
0	0.000	0.00607	–
15	0.649	0.00006	0.2E-07

values of α to agree, since they are coefficients for different sets of piecewise polynomials. The real tests are in the shape of the bump, and the functional match.

From Figures 15.1 and 15.2, it's easy to see that the piecewise linear bump is "trying" to match the shape of the piecewise quadratic, but it's nearly impossible to see and compare the horizontal velocities along the profile line. To rectify that problem, in Figure 15.3 we plot the horizontal velocity functions for the target and optimizing flows, along the profile line. The target values are drawn with a dashed line, but the match is so good between the two functions that it's hard to see.

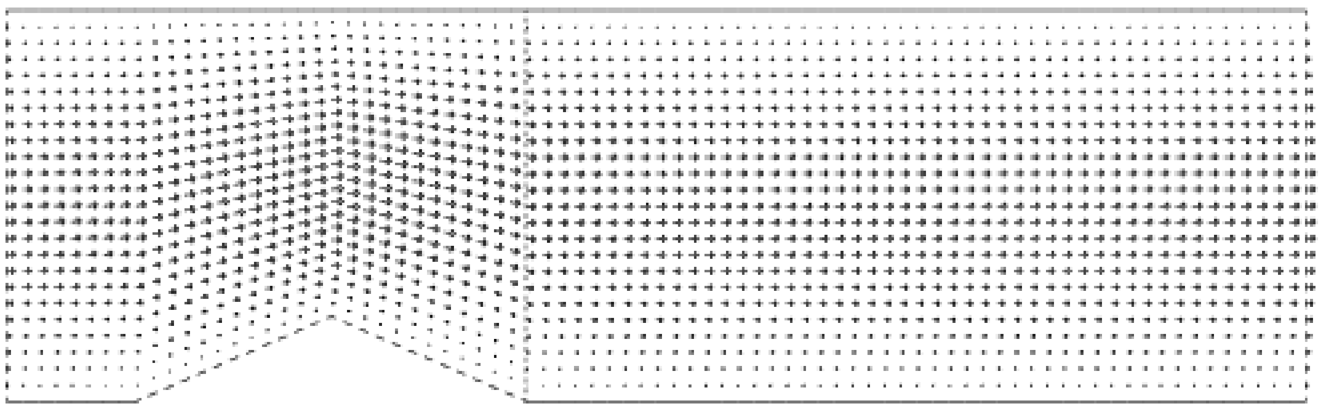

Figure 15.2: Example 1: The optimizing flow.
Only piecewise linear bumps are feasible.

15.3 Example 2: Differing Three-Parameter Bump Shapes

After the results of our first example, we are interested in pushing the problem a little further. We will essentially repeat the calculation, but with three parameters used to represent the bumps. Also, we will make the target bump higher, using the values $\alpha = (0.75, 1.0, 0.75)$.

From the optimization history in Table 15.2, it seems reasonable to believe that the minimal cost, and the values of the second and third bump parameters have been well approximated. Looking at the behavior of the bump parameters, it seems as though they are determined from right to left, with the bump parameter closest to the sampling line presumably having the biggest influence, and hence being determined first. We will accept our results, while realizing that there may be some uncertainty in the value of α_1.

From Figures 15.4 and 15.5, it's easy to see that the bumps are dissimilar in shape. The piecewise linear bump is by no means the interpolant of the quadratic bump, nor does it seem close to one.

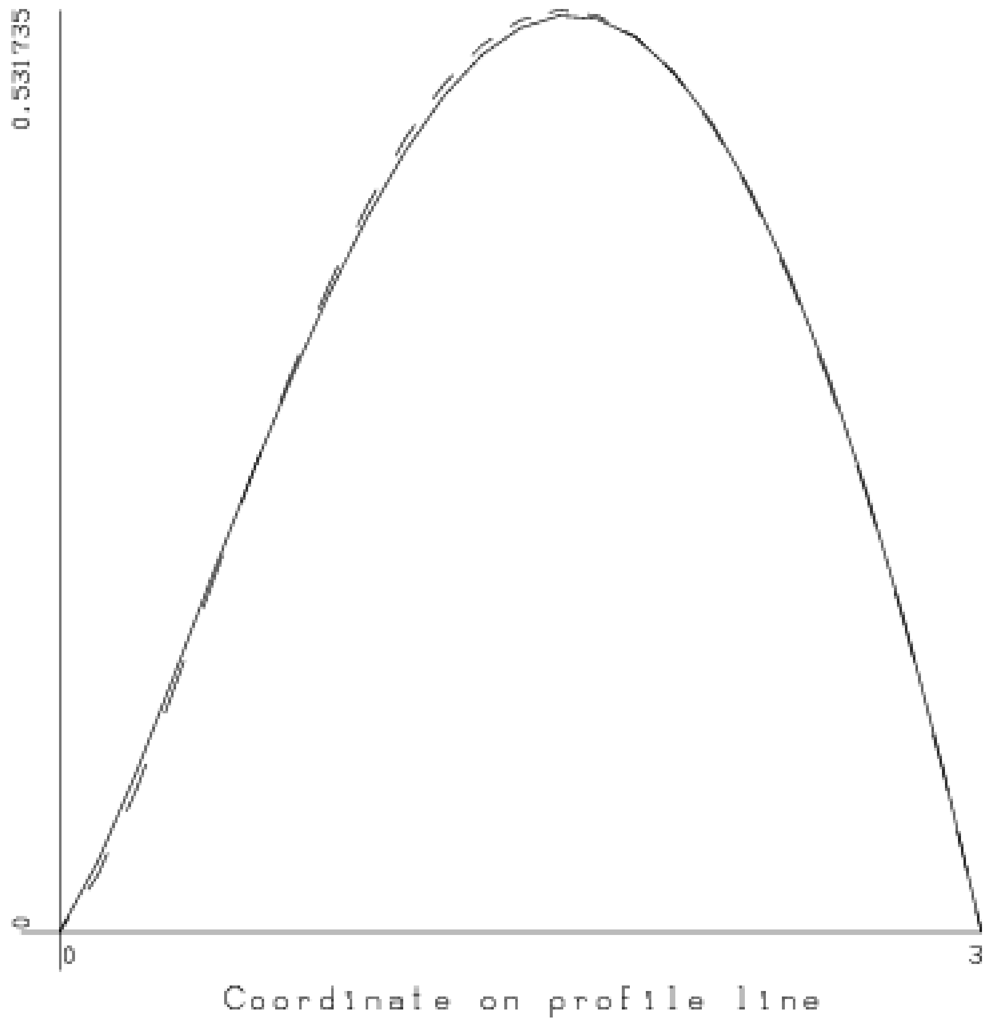

Figure 15.3: Example 1: Comparison of target and achieved flows.

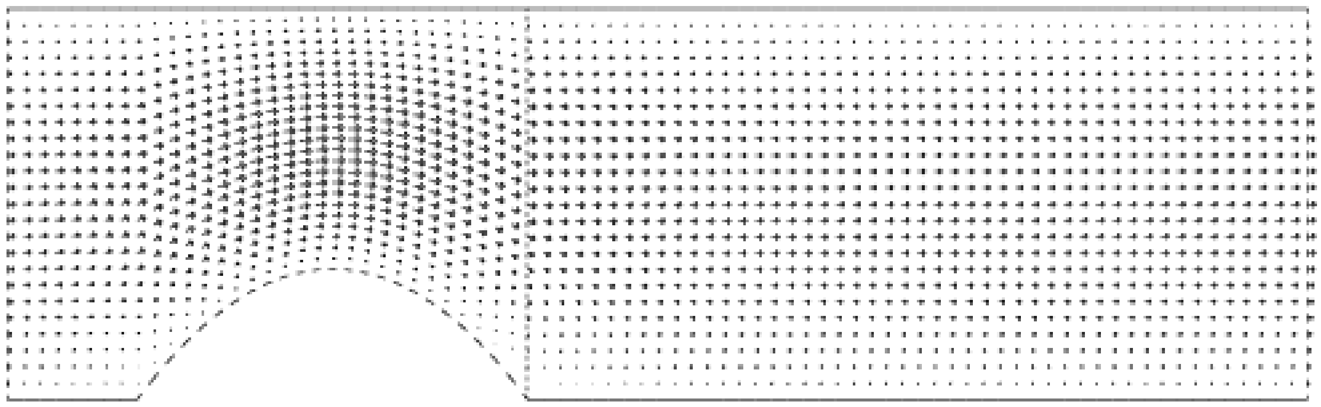

Figure 15.4: Example 2: The target flow.
A piecewise quadratic bump is used.

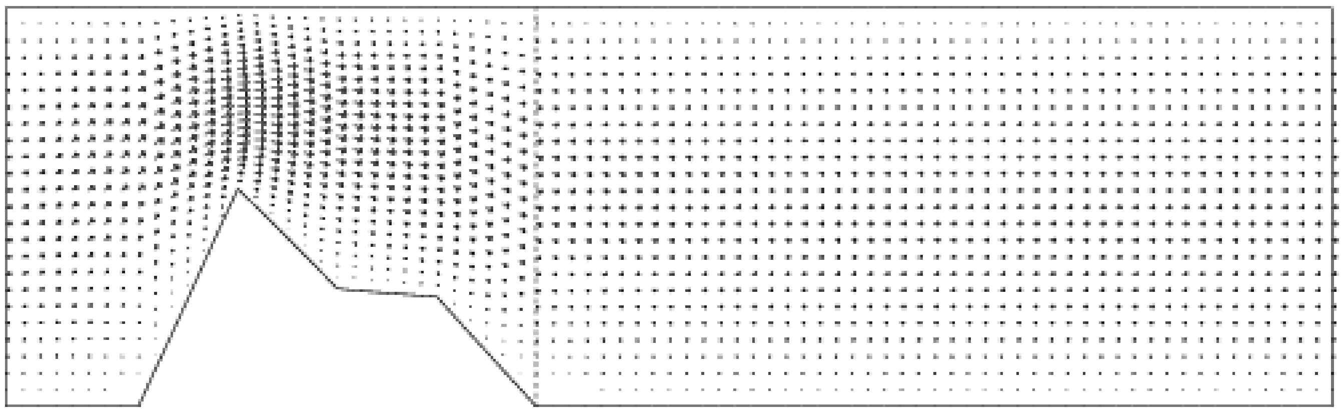

Figure 15.5: Example 2: The optimizing flow.
Only piecewise linear bumps are feasible.

Table 15.2: Example 2: Optimization results.

Steps	α_1	α_2	α_3	$\mathcal{J}_2^h$	$\|\|\nabla \mathcal{J}_2^h\|\|_\infty$
0	0.000	0.000	0.000	0.343E-01	–
10	0.033	0.175	0.845	0.266E-03	–
20	0.151	0.666	0.859	0.179E-03	–
30	0.230	0.996	0.794	0.591E-05	–
40	0.231	1.000	0.791	0.575E-05	–
50	0.232	1.000	0.792	0.575E-05	–
60	0.471	1.006	0.793	0.574E-05	–
70	0.595	1.009	0.794	0.572E-05	–
80	0.599	1.010	0.793	0.571E-05	–
90	0.995	1.022	0.798	0.544E-05	–
100	1.161	1.006	0.805	0.515E-05	–
110	1.246	0.999	0.806	0.488E-05	–
120	1.364	0.988	0.812	0.458E-05	–
130	1.638	0.886	0.823	0.378E-05	–
135	1.638	0.885	0.823	0.378E-05	0.2E-08

Of course, since the target isn't feasible, we can't judge our results by how well the bumps match, but should instead refer to the match of horizontal velocities along the sampling line. In our plots which show the entire region, this is very difficult to see. To rectify that problem, in Figure 15.6 we plot the horizontal velocity functions for the target and optimizing flows, along the profile line. The target values are drawn with a dashed line, but even now the discrepancy between the two is difficult to spot.

It may be of interest to compute the flow solution generated by the linear interpolant to the quadratic bump, which would have been a natural guess for the minimizing parameters. The results are shown in Figure 15.7, where the discrepancy between the target and computed flows is clear. The value of $\mathcal{J}(\alpha)$ is 0.1E-03, which certainly shows that our minimizing solution has done better.

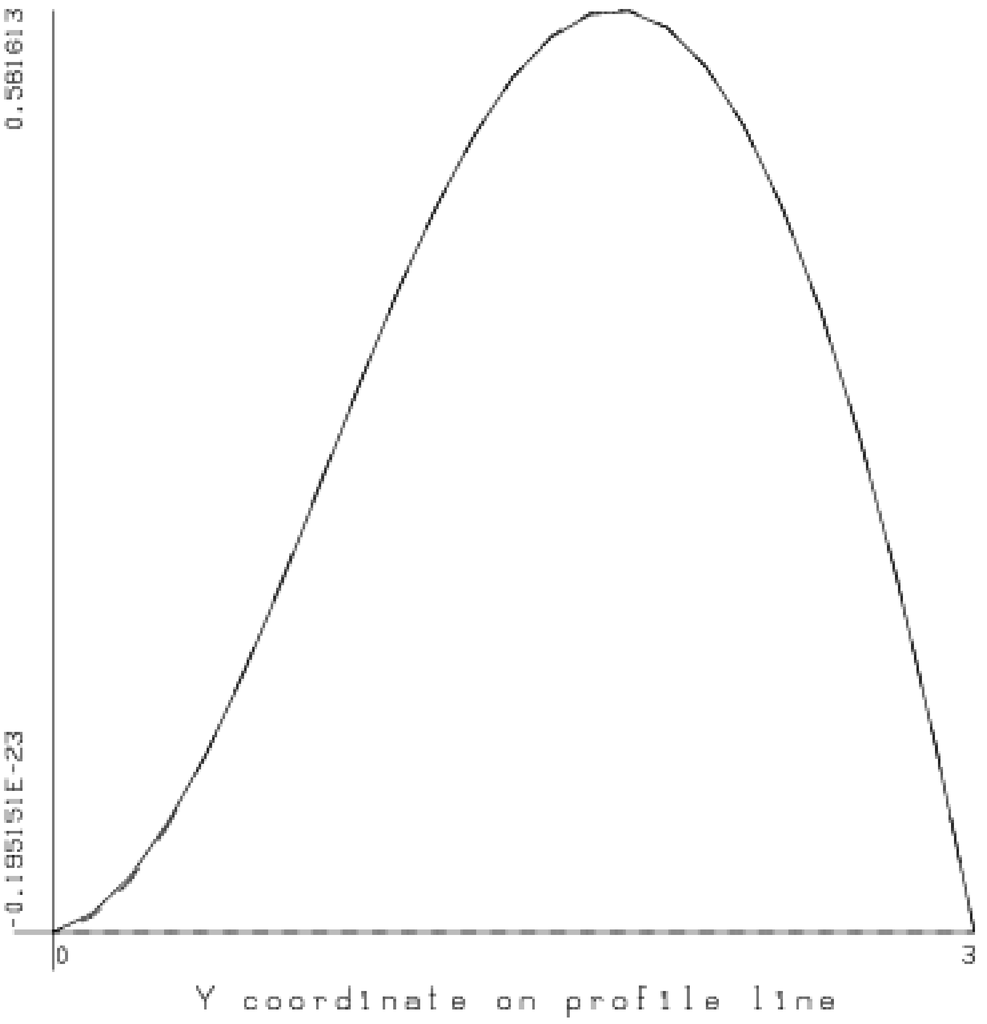

Figure 15.6: Example 2: Comparison of the target and minimizing flows.

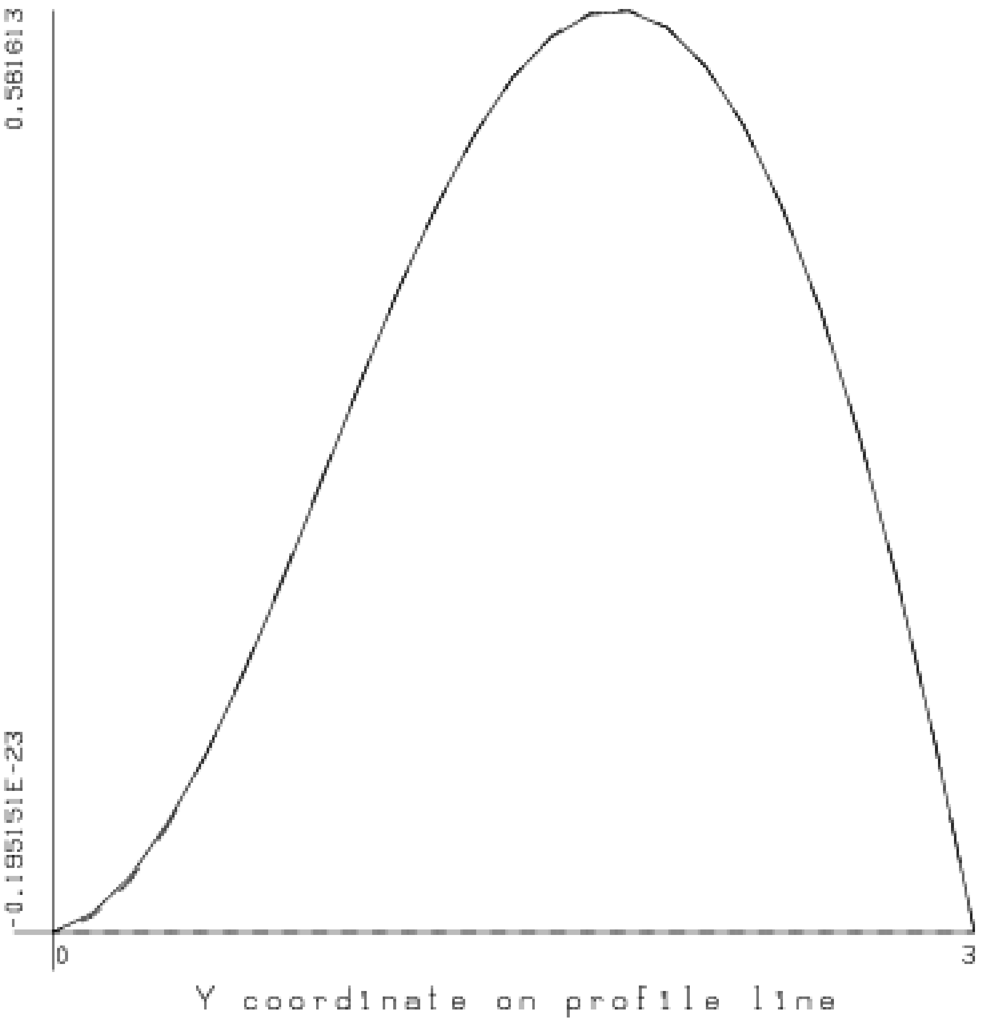

Figure 15.7: Example 2: Comparison of the target and linear interpolant flows.

Table 15.3: Example 3: Optimization results.

Steps	λ_1	λ_2	λ_3	$\mathcal{J}_2^h$	$\|\|\nabla \mathcal{J}_2^h\|\|_\infty$
0	0.1000	0.1000	0.1000	5.974	–
22	1.9608	-2.0536	2.9955	0.128E-01	0.9E-08

15.4 Example 3: Three-Parameter Inflows

In our next example, we leave the bump fixed, but allow the inflow to vary. We model the shape of the inflow by three parameters. Such a varying inflow might be generated by using several separate fans or deflectors in a wind tunnel.

We generate an arbitrary curve for the horizontal flow at the profile line, using the formula:

$$\begin{aligned} u_1(x_s, y) &= (3-y)y/m_1 \\ u_2(x_s, y) &= (3-y)y(1.5-y)/m_2 \\ u^T(x_s, y) &= \gamma_1 u_1(x_s, y) + \gamma_2 u_2(x_s, y) \end{aligned}$$

where m_1 is chosen so that the maximum value of the parabolic flow u_1 over $[0, 3]$ is 1, and m_2 is chosen similarly for u_2. The quantities γ_1 and γ_2 are chosen by the user at run-time. Since parabolic flow is the natural tendency for our problem, we may think of u_2 as a perturbation to this natural tendency. The larger γ_2 is, relative to γ_1, the more "unnatural" the target flow will be, and the harder to achieve. For our demonstration problem, we choose $\gamma_1 = 2$, $\gamma_2 = -0.5$. Figure 15.8 is a graph of the resulting target profile.

Now the value of the target flow along the profile is all we need to define an optimization problem, so we can proceed to solve this problem, using a Reynolds number $Re = 10$. Because the target flow was not generated in the usual way, we expect that our optimizing flow will not achieve a zero cost functional, although we do want the cost gradients to be close to zero. A summary of our results are in Table 15.3.

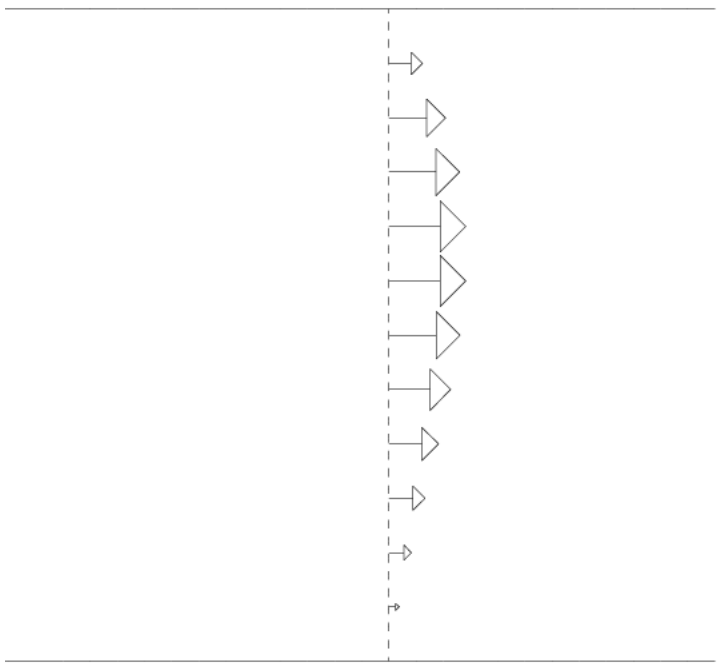

Figure 15.8: Example 3: The “artificial” flow.
Values $\gamma_1 = 2.0$, $\gamma_2 = -0.5$ were used.

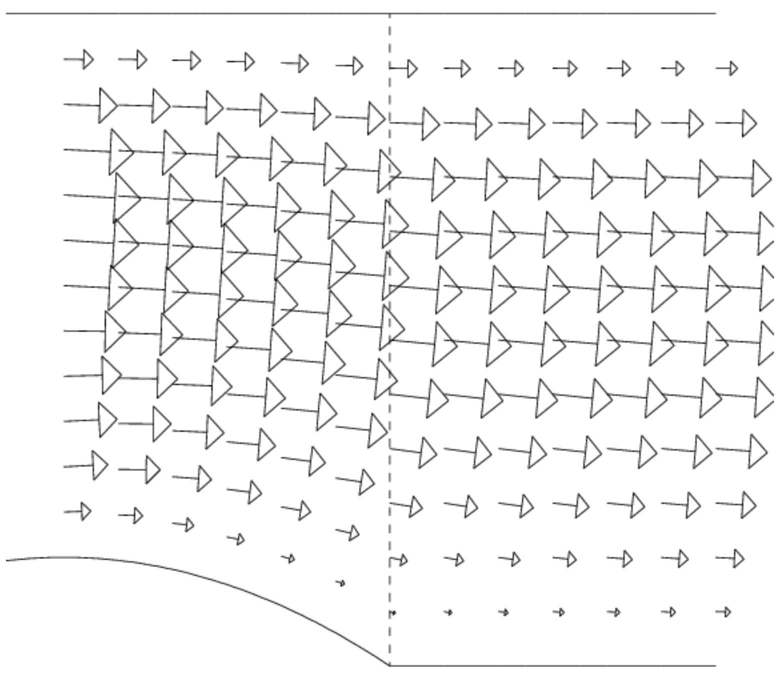

Figure 15.9: Example 3: The optimizing flow, near the profile line.

In Figure 15.9 we show the optimizing velocity field near the profile line, and in Figure 15.10 the optimizing velocity field at the inflow, where it is plain that a certain amount of *outflow* is generated. Finally, in Figure 15.11 we show the excellent match between the target and optimizing velocities.

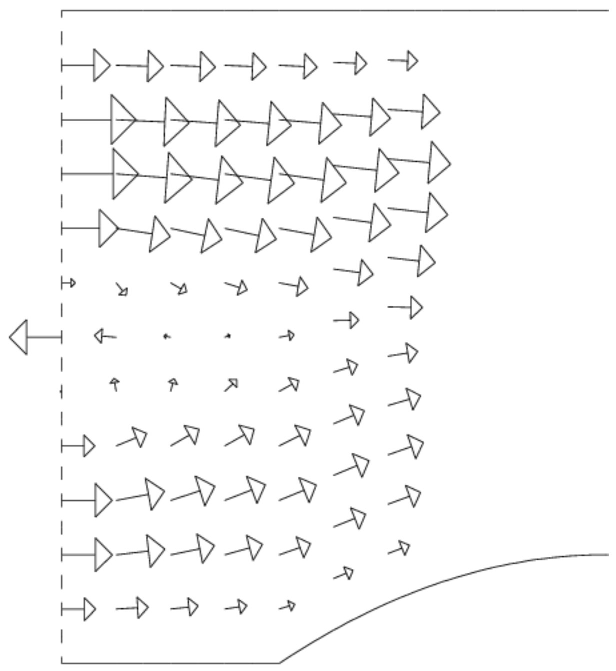

Figure 15.10: Example 3: The optimizing inflow (includes outflow!).

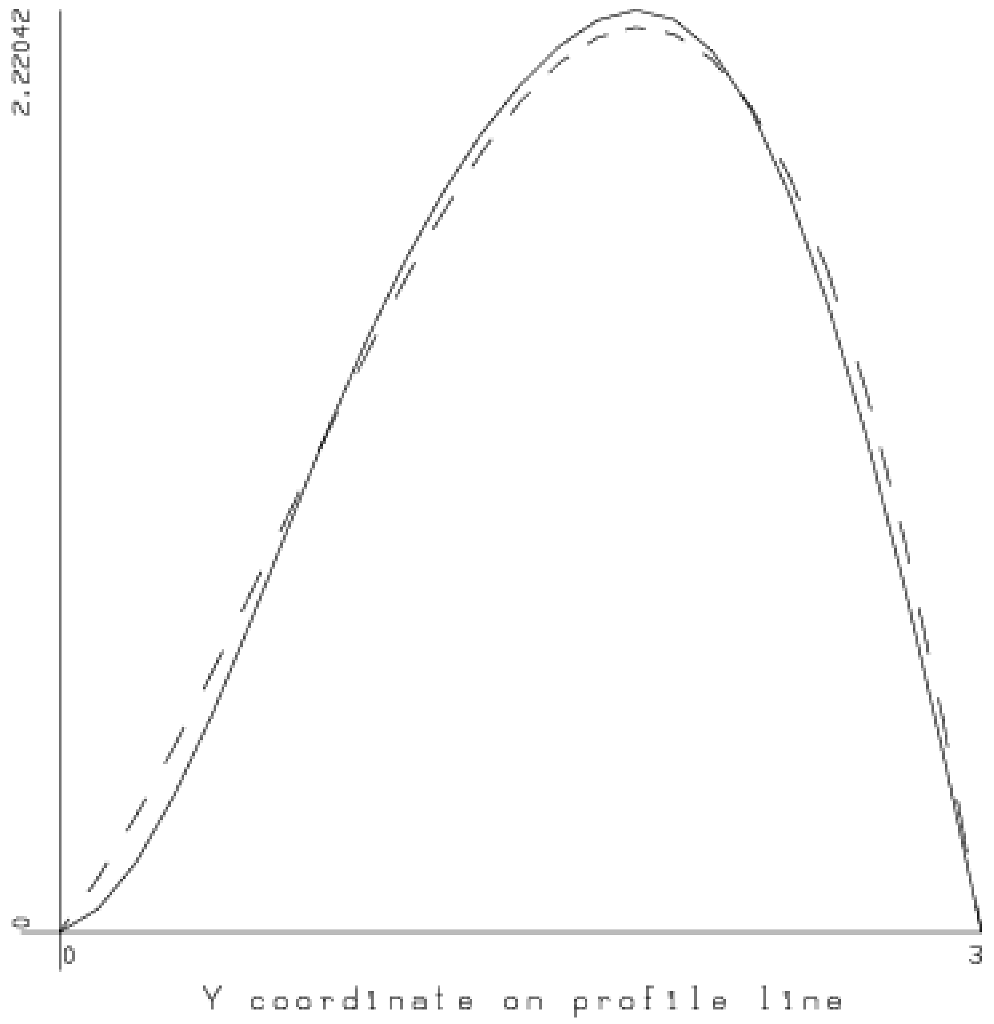

Figure 15.11: Example 3: The match between target and desired flows.

Chapter 16

EFFICIENT SOLUTION OF THE OPTIMIZATION PROBLEM

16.1 Introduction

The optimization of a cost functional associated with a Navier Stokes problem is *expensive*; that is, the usual solution algorithm requires a considerable amount of arithmetic operations and computer time. It is important to investigate whether there are simple techniques for reducing the computational expense.

We choose a standard problem and estimate the amount of work that will be required to carry out an optimization using our original algorithm. We then consider several variations of this algorithm that strive to reduce the amount of work involved.

We also investigate how the computational expense will grow as the Reynolds number is increased, which makes the nonlinear terms more prominent, and the problem harder to solve.

16.2 A Standard Test Problem and Algorithm

To investigate methods for speeding up the optimization, we will find it useful to pick a standard problem, carry out the optimization procedure, and carefully record the computational effort required, paying particular attention to the assembly and factorization of the Jacobian matrix.

As a standard problem, we take a four parameter case, with one inflow parameter and three bump parameters. We fix the *Re* parameter at 1. Target data is generated using the parameter values $\lambda^T = 0.5$, $\alpha^T = (0.375, 0.5, 0.375)$.

For the discretized problem, we will use a mesh parameter $h = 0.125$, resulting in 960 elements, 2025 nodes, and 4190 unknowns. The Jacobian matrix is banded, with a bandwidth of 225 entries.

For the low Reynolds numbers we will be examining, it is usually not necessary to employ Picard iteration. Therefore, we will usually rely exclusively on Newton's method, using Picard iteration only as a fallback. We will use a Newton convergence tolerance of **TolNew**=1.0E-9.

The optimization will be carried out with an optimization tolerance of **TolOpt**=1.0E-9. We will start from the usual zero initial estimate, which has a cost functional value of 0.436.

To carry out the computation, let us begin with a simple-minded approach, which we will call *Algorithm 1.* Algorithm 1 will re-evaluate and refactor the Jacobian on every step of the Newton process. Once the main solution point is computed, finite coefficient differences will be made to get approximations of the sensitivities. The proper computation of these finite difference points will require yet another Newton process for each parameter component; we will update the Jacobian during these subsidiary Newton processes as well.

If we apply this algorithm to our problem, 34 optimization steps are required, after which the optimization code declares satisfactory convergence to the parameters $\lambda = 0.499999$, $\alpha = (0.374365, 0.500065, 0.375045)$. At this point, the cost functional $\mathcal{J}_2^h(\lambda, \alpha) = 0.363E-9$ and the estimated gradients are $\dfrac{\partial \mathcal{J}_2^h}{\partial \lambda} = 0.6E-05$ and $\dfrac{\partial \mathcal{J}_2^h}{\partial \alpha} = (-0.2E-06, 0.4E-05, 0.6E-05)$. During this run we computed flow solutions at 169 separate sets of parameter values, which in turn required a total of 187 Newton iteration steps, as well as 187 Jacobian evaluations and factorizations. The optimization of our standard problem required 1112 seconds of CPU time on a DEC Alpha.

16.3 Expensive Computational Tasks

A timing analysis of the program reveals that the most expensive operations are:

- The evaluation of the finite element basis functions;
- The evaluation and factorization of the Jacobian;
- The solution of linear systems.

with the Jacobian factorization being the single most expensive item.

We have already attempted to economize on the basis function evaluations; we precompute the values at all quadrature points, and then simply look them up when needed. However, if there are geometric parameters, then for each new set of parameters these quantities must be recomputed, reducing our efficiency gain. No further improvement is likely here, since the number of evaluations on the finite element basis functions depends in turn on the number of evaluations of the Jacobian.

Our Jacobian is symmetrically banded. We apply the LAPACK banded factorization routine DGBTRF to compute its LU factors. The amount of work required for this operation is

Table 16.1: Matrix work required for various values of h. Work is measured in millions of floating operations.

h	MBand	Neqn	Factor (MegaOps)	Solve (MegaOps)
0.2500	66	1166	10	0.2
0.1666	93	2555	44	0.7
0.1250	120	4484	129	1.6
0.0833	174	9962	603	5.2
0.0625	228	17600	1829	12.0

roughly $O(2 * Neqn * Mband^2)$, where $Neqn$ is the number of equations represented by the matrix, and $Mband$ is the half bandwidth.

To solve a particular linear system, we use the corresponding LAPACK routine DGBTRS. The cost for a single linear solution from this routine is roughly $O(3 * Neqn * Mband)$.

The number of equations and the bandwidth depend, in turn, on the mesh parameter h. As the mesh parameter is halved, for instance, we expect roughly that $Mband$ will double, and $Neqn$ quadruple. This means that the computational work goes up by a factor of 16. If we halve h twice, it will take us roughly 256 times as long to factor the corresponding Jacobian. No other operation in the program will have this explosive growth with decreasing h. Therefore, as h decreases, it is vital that we be able to reduce the number of Jacobians we need. In some cases, reducing the number of Jacobians will increase the number of linear solutions required.

Table 16.1 lists the amount of work required for factorizations and solutions for a sequence of values of h. We can see, for instance, that in going from $h = 0.25$ to $h = 0.0625$, our estimate of a 256-fold increase in factorization work is reasonably close to the actual 182-fold increase. From this table, we can see that it takes quite a large number of linear solutions to equal the cost of one factorization.

Of course, there are many hidden costs that this table cannot show. For instance, we have not measured the costs of *evaluating* the Jacobian, which is roughly equal to the cost of evaluating the Newton residual. To make a more accurate assessment of, say, the relative cost of one Jacobian factorization versus several Newton steps, we would need to measure the evaluation work as well.

However, this table has given us a clear indication of the large amount of work required for Jacobian factorization, and its explosive growth as h decreases.

16.4 A Comparison of Algorithms at $Re = 1$

Since the evaluation and factorization of the Jacobian is our most costly computational task, we will now investigate whether we can reduce the overall cost of the computation by trying to reduce the amount of work devoted to the Jacobian.

The simplest change we can make to Algorithm 1 is to hold the Jacobian fixed when computing flow solutions required as part of a finite difference calculation. This is surely a reasonable economy; the last Jacobian calculated in the Newton iteration for a main solution point is likely to be quite close to the Jacobian at any of the finite difference points, whose parameters differ in a minuscule way from those of the main solution point. We will call this method *Algorithm 2.*

In fact, during a Newton iteration, the iterates are all likely to be close enough so that we could actually use a single Jacobian to compute several or all of the Newton steps. For our purposes, we will assume that at most 5 Newton steps in a row are computed with a fixed Jacobian, and that the Jacobian is *always* reevaluated and factored at the beginning of each Newton process. As before, we will hold the Jacobian fixed for Newton processes associated with finite difference computations. We call this method *Algorithm 3.*

Table 16.2: Work carried out for various algorithms, $Re = 1$.
Pure Newton algorithm, tolerance **TolNew**=1.0E-9.

Algorithm	Opt Steps	Flow Solves	Newton Steps	Matrix Factors	CPU (sec)
1	34	169	187	187	1112
2	34	169	187	55	450
3	33	164	187	33	337
4	37	184	608	6	424
5	35	38	73	73	498
6	34	37	90	34	310
7	36	179	436	13	400

To see how far we can reduce the factorizations of the Jacobian, we also consider *Algorithm 4*, which *never* evaluates the Jacobian except on the very first Newton process, or if the Newton process has failed to converge. *Algorithm 7* is similar, but instead of waiting for the Newton process to fail, we evaluate the Jacobian and then reuse it for the entire Newton process, and for all Newton processes associated with "nearby" parameter sets.

Algorithm 5 is similar to Algorithm 2, except that we use discretized sensitivities instead of finite coefficient differences. *Algorithm 6* is the same as Algorithm 3, except that we use discretized sensitivities instead of finite differences.

We applied these algorithms to our standard problem, and display the results in Table 16.2.

What may we conclude from these results? At the moment, we wish primarily to consider questions of efficiency. If we consider the CPU measurement to be a reasonable measure of the overall work performed by the algorithm, then we may take our task to be the explanation of the timing results in terms of the underlying algorithms.

The first remarks then go to the longest run by far, our basic algorithm. We know that matrix factorization is the most expensive single operation in our algorithm; this correlates well with the very high number of factorizations carried out by Algorithm 1.

Surely the number of factorizations cannot be the only factor, else Algorithm 4 would have been our quickest, with only 5. The fact that Algorithms 3 and 6, with more matrix factorizations, are substantially faster, means that there are other sources of work that must also be controlled. Indeed, we can note that Algorithm 4 paid for its very low factorization rate with a correspondingly high number of Newton iterations. Each Jacobian evaluation, for instance, was preceded immediately by a Newton iteration that failed to converge after taking the maximum number of steps, accounting for a total of 200 steps. Since each Newton step requires the solution of a linear system, which is by no means a cheap process, we can attribute the mediocre performance of Algorithm 4 to a failure to control the number of linear system solves.

Algorithm 7, while similar to #4, manages to reduce the number of Newton steps by about 200, at a slight cost of 7 extra Jacobian evaluations. However, there is little improvement in speed for this problem.

Now let us consider Algorithm 6, which produced the lowest time to solve the problem. It is reasonable to attribute its speed to the combination of a low number of matrix factorizations and a very low number of Newton iterations. Algorithm 6 has limits on its stinginess: it reuses the Jacobian, but no more than 4 times. This seems to guarantee good Newton convergence behavior at a significantly lower cost in factorizations.

We note that the second best timing was for Algorithm 3, which actually had a high number of flow solves and Newton steps (because it used finite differences instead of sensitivities). Apparently, the fact that it was able to keep the number of factorizations almost as low as for Algorithm 6 was the decisive factor. The rule seems to be that factorizations are very important; linear solves are only important if their number gets “out of hand”.

So far, our results at $Re = 1$ have suggested that we need to be aware of an appropriate

Table 16.3: Work carried out for various algorithms, $Re = 100$.

Algorithm	Opt Steps	Flow Solves	Newton Steps	Matrix Factors	CPU (sec)
1	36	180	218	218	6189
2	38	190	228	80	2639
3	37	185	292	43	1813
4	38	190	1029	9	1912
5	36	40	105	105	3146
6	49	58	292	96	3298
7	39	195	702	11	1514

error tolerance for the Newton iteration and that we need to use the Newton system matrix wisely, neither updating it every step, nor reusing it to the point where the Newton process is severely hampered.

16.5 A Comparison of Algorithms at $Re = 100$

Our efforts at improving the efficiency of the algorithm do not show as much profit as we might like. But this may be that the flow problem is too easy to solve when the Reynolds number is low. As *Re* increases, the problem becomes more nonlinear, the Newton process requires more steps, and conservation of Jacobian evaluations will provide a bigger payoff. Let us recompute our results using the same set of algorithms, but for a Reynolds number of $Re = 100$, and a mesh spacing of $h = 0.083$. Table 16.3 shows the new timings.

Here we see a dramatic separation of the algorithms. Algorithm 1 is, as expected, the most expensive. But now we can rate Algorithms 2, 5, and 6 as mediocre, Algorithms 3 and 4 as good, and Algorithm 7 as superior.

We can find ready explanations for some of the results in the table. Our mesh parameter h had to decrease because the Reynolds number had increased. The overall cost for each algorithm is then multiplied by a factor of 4 to 10. The smaller mesh spacing means more variables, and hence a bigger Jacobian. The cost of evaluating and factoring the Jacobian matrix becomes relatively more important, and hence, algorithms that conserve on this single cost will do better.

The fact that Algorithm 7 did better than Algorithm 4 emphasizes the importance of "preventive" Jacobian updating. It can be a bad idea to wait until the Jacobian is absolutely useless before updating it; a very poor Jacobian may be usable, but very ineffective. Algorithm 7's technique of updating the Jacobian when the parameters have changed significantly seems to pay off. Good, fresh Jacobian information is needed repeatedly in the beginning of an optimization, but towards the end, when the iterates cluster around a single value, there is much less need to reevaluate.

Algorithm 3, which updates the Jacobian on each Newton process, but then holds it fixed, should be compared to Algorithm 2, which updates the Jacobian on every Newton step. Algorithm 3 cut down the number of Jacobian evaluations by roughly half, and was able to cut 800 seconds off its CPU time.

It is surprising that Algorithm 6 did so poorly. It was the best performer for $Re = 1$. However, it had a failure of the Newton process early in the computation, which cost it 60 wasted Newton steps, and it then seems to have had optimization difficulties, because of the use of discretized sensitivities, that made it compute 49 optimization steps, much above the range of 36 to 39 required by the other algorithms.

16.6 Appropriate Newton Tolerances

Since the Newton iterations are our main expense, it might seem that an easy way to cut down on work would be to loosen the Newton tolerance **TolNew** used for determining convergence. It is surely true that this can have the effect of shortening any one Newton process. However loosening the Newton tolerance means increasing the inaccuracy that we will accept in our solution; this same inaccuracy then affects our calculations of the cost function, the sensitivities, and the cost gradients. Inaccurate cost data can greatly prolong the optimization process, particularly because descent directions will be inaccurate. And once the inaccuracies in the computed cost functional become large enough, the formerly smooth graph of the cost functional will behave as though it were covered by spurious local minima, bringing the optimization process to a confused and incorrect halt.

Normally, one would simply choose the Newton tolerance to be a "small" value. We have typically used values of the order of 1.0E-9 or 1.0E-10. However, working on higher Reynolds number problems, we considered using a tolerance of 1.0E-7. We were surprised at certain unexpected behaviors in the optimization process, and so we returned to a lower Reynolds number where we had more experience and did some comparisons.

As a model problem for this study, let us take a four parameter problem, with one inflow parameter λ and three bump parameters α, with target parameters $\lambda^T = 0.5$ and $\alpha^T = (0.375, 0.5, 0.375)$ and Re fixed at 1. A mesh parameter of $h = 0.125$ was used. Let us use adjusted finite coefficient differences as estimates of the discrete sensitivities used to compute the cost functional gradient. (Note that we don't update the Jacobian for Newton steps carried out on finite difference computations.) Then, if we vary the Newton tolerance **TolNew**, leaving all other quantities fixed, we get the work estimates listed in Table 16.4.

The table indicates that an acceptable solution can be computed for **TolNew**=1.0E-9 or less. The most efficient calculation occurs at **TolNew**=1.0E-10, where, it seems the cost of accuracy in the Newton solution has been roughly balanced by the benefits of a rapid, accurate optimization. As the tolerance grows above this range, the quality of the optimization deteriorates very rapidly, because the data is too poor to work with. If instead, we decrease the tolerance, the work slowly but relentlessly rises, as we waste time computing

Table 16.4: Work carried out for various Newton tolerances, $Re = 1$.
"PErr" is the L2 parameter error between the computed and global minimizers.

TolNew	Opt Steps	Flow Solves	Newton Steps	Matrix Factors	CPU (sec)	PErr
1.0E-05	63	314	314	69	470	0.44
1.0E-06	40	199	192	36	265	0.23
1.0E-07	31	154	152	32	221	0.26
1.0E-08	36	179	193	52	318	0.0004
1.0E-09	34	169	187	55	316	0.001
1.0E-10	33	164	183	55	305	0.001
1.0E-11	34	169	193	61	330	0.001
1.0E-12	34	169	197	65	346	0.001
1.0E-13	34	169	198	66	349	0.001
1.0E-14	34	169	301	70	425	0.001
1.0E-15	34	169	335	77	471	0.001

over-accurate solutions. For a tolerance of **TolNew**=1.0E-16, the entire optimization fails because the Newton iteration cannot converge to the given tolerance.

Note, however, what a very simple problem this must be. Up until **TolNew**=1.0E-14, the number of flow solutions is roughly equal to the number of Newton steps. In other words, a single Newton step is enough to correct the initial guess to the desired tolerance.

We repeat the calculations for a higher Reynolds value of 100, with the results shown in Table 16.5. Certain features of our previous results persist. In particular, we see again that the overall CPU time suddenly increases if the tolerances are too loose, while the CPU time slowly rises as we decrease the tolerances. Again, there seems to be a discontinuous behavior in the estimation of the parameters. Above a certain tolerance, the approximation is very poor. Below it, the approximation seems to be about the same, regardless of the tolerance used; only the work increases.

When comparing results at $Re = 1$ and $Re = 100$, we notice that the parameters for the global minimizer are approximated better for $Re = 100$, by about a factor of 10, and that

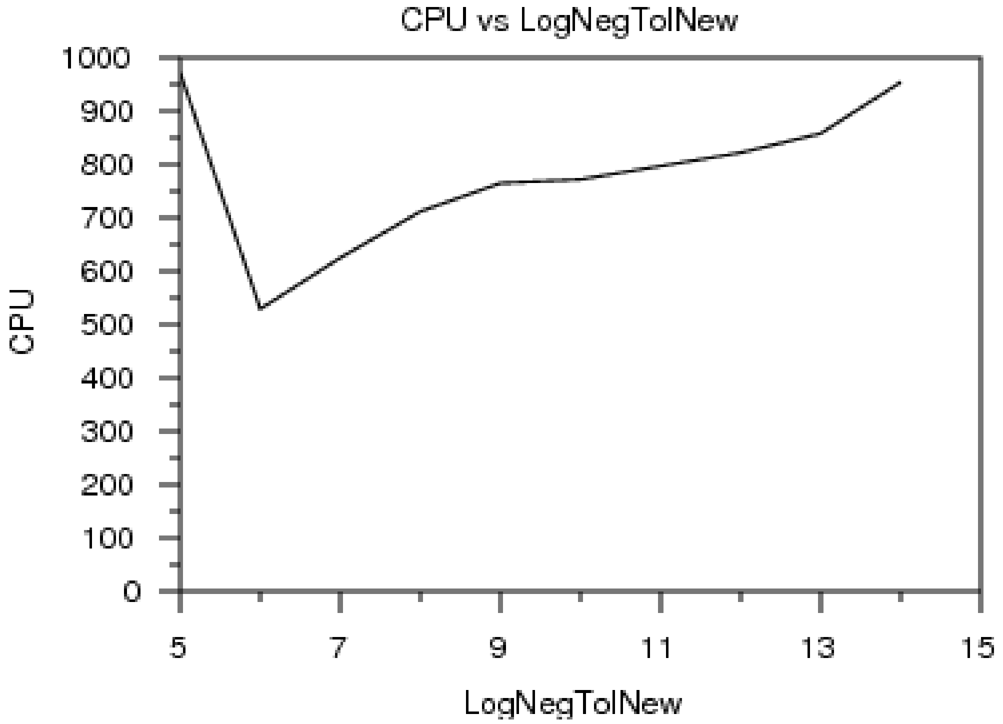

Figure 16.1: CPU time as a function of Newton tolerance, $Re = 1$.

this better approximation begins for a higher tolerance. On the other hand, the amount of work required to optimize the $Re = 100$ problem begins to rise at a much higher Newton tolerance, suggesting that the gradients are well approximated already. This suggests that while the higher Reynolds number flows are harder to solve *as flow problems*, they are easier to solve *as optimization problems*. In other words, in a high speed flow, small differences in the shape of the obstacle make a bigger difference in the downstream flow.

During a particular optimization, it may not at first be clear what an appropriate Newton tolerance should be. Fortunately, the signs of an inappropriate Newton tolerance can be easy to spot: an optimization that seems to drift aimlessly once the cost functional or gradient values have dropped small enough, or a situation where the Newton iteration repeatedly fails to reach the requested tolerance.

Table 16.5: Work carried out for various Newton tolerances, $Re = 100$.
"PErr" is the L2 parameter error between the computed and global minimizers.

TolNew	Opt Steps	Flow Solves	Newton Steps	Matrix Factors	CPU (sec)	PErr
1.0E-05	64	316	350	112	971	0.18
1.0E-06	38	186	193	59	529	0.44
1.0E-07	42	210	234	70	625	0.0002
1.0E-08	43	215	252	84	712	0.0002
1.0E-09	42	210	257	93	765	0.0005
1.0E-10	41	205	255	95	772	0.0001
1.0E-11	41	205	259	99	797	0.0001
1.0E-12	41	205	263	103	822	0.0001
1.0E-13	41	205	307	105	858	0.0001
1.0E-14	41	205	428	112	954	0.0002

Chapter 17

THE COMPUTATIONAL ALGORITHM

17.1 Introduction

In this chapter, we look in detail at the algorithm used to produce the results discussed in this thesis. We first give a logical outline of the order of operations in the algorithm. We next discuss the names and meaning of some of the user input quantities. We then describe the individual steps of the algorithm in terms of the variables that are used and altered at each step.

Finally, we discuss our choice of optimization software, including the algorithm employed, the input required from the user, and the types of stopping criteria.

17.2 Black Box Versus One-Shot Methods

The strategy we employ for our flow optimization is an example of the *black box* method. We may consider our overall algorithm to comprise two sub-algorithms: the flow solver, and the optimization code. The term "black box" is meant to suggest that the details of the flow solving are not used in any way by the optimization code. We may summarize our algorithm as repeated executions of:

$$\begin{aligned}
&\text{Given parameters } \beta : && (17.1)\\
&\text{Solve for state variables } F(u^h(\beta), v^h(\beta), p^h(\beta), \beta) = 0 && (17.2)\\
&\text{Evaluate the cost} \mathcal{J}(\beta) = J(u^h(\beta), v^h(\beta), p^h(\beta), \beta) && (17.3)\\
&\text{Accept } \beta\text{, or terminate, or repeat.} && (17.4)
\end{aligned}$$

The function F is meant to represent the Navier Stokes system.

To understand that this is not the only possible approach, one might consider the following method, which seeks to solve the same problem, though using the cruder optimization technique of seeking a zero of the gradient of J.

$$\begin{aligned}
\text{Given } (u^h_0, v^h_0, p^h_0, \beta_0) & & & (17.5)\\
\text{Find } (u^h, v^h, p^h, \beta) \text{ satisfying the pair of equations} & & & (17.6)\\
F(u^h, v^h, p^h, \beta) &= 0 & & (17.7)\\
\nabla J(u^h, v^h, p^h, \beta) &= 0 & & (17.8)
\end{aligned}$$

where J is the original cost functional that is written in terms of the flow functions as well as the parameters. Notice that the flow functions u, v and p used as arguments of J are *not* required beforehand to be actual flow solutions. This method would start with the arbitrary set of flow functions and parameters $(u^h_0, v^h_0, p^h_0, \beta_0)$, and try to solve $F = 0$ and $\nabla J = 0$ simultaneously. Thus, the optimization and flow portions of the calculation would be done *simultaneously.* Such an approach is called a *one-shot method.*

17.3 Black Box Flow Optimization Algorithm

Our algorithm carries out some initializations, and then repeatedly executes an optimization loop. It begins the loop with some set of parameter values, finds the corresponding flow solution, evaluates the cost functional and the approximate cost gradient, and reports to the optimization code for further instructions. The simplified version of our algorithm that is described here assumes that our cost functional depends only on the horizontal velocity discrepancy, that the cost functional target data is computed in the usual way as sampled data of a flow generated by target parameters, and that discretized sensitivities are used to estimate the cost gradient.

Algorithm 17.1 (Black Box Flow Optimization)

```
Get user input.
Compute Utar, the target horizontal velocities.
Initialize optimal parameter estimate Para to zero.
Begin optimization loop:
    Estimate flow solution (U, V, P) for current parameters Para.
    Use Picard iteration to "improve" (U, V, P) for use by Newton.
    Use Newton iteration to produce correct flow solution (U, V, P).
    Compute discretized flow sensitivities dUdP at Para.
    Compute $\mathbf{Cost} = \int(\mathbf{Utar}(\mathbf{XProf}, y) - \mathbf{U}(\mathbf{XProf}, y))\, dy$
    If finite difference sensitivities are desired, find nearby flow solutions.
    Compute cost gradient vector dCdP.
    Report current values of Para, Cost and dCdP to optimization code.
    Optimization code makes one of the following recommendations:
        Current parameters acceptable. Stop with success.
        Optimization must be abandoned. Stop with failure.
        New estimate of parameters in Para. Repeat optimization loop
End optimization loop
```

In the succeeding sections, we discuss portions of this algorithm in detail.

17.4 User Input

The program accepts at runtime a file of user input. This allows the user to specify information about the following matters:

- the physical problem to be solved;
- the flow region discretization;
- the cost functional;
- starting parameter values;
- the Newton iteration;
- the cost gradient calculation;
- the optimization.

We will now list the exact information that may be specified by the user.

17.4.1 Physical Problem Input

The basic flow region is assumed to be a rectangle. The user gives the dimensions of this rectangle as follows:

XLngth	The length of the flow region.
YLngth	The height of the flow region.

The number of parameters used must be specified by the user:

NParB	The number of bump parameters.
NParF	The number of inflow parameters.

Note that there are a total of **NPar=NParB+NParF+1** parameters, because the Reynolds number *Re* is always considered a parameter, although it need not be varied in a particular problem.

17.4.2 Discretization Input

The flow region is covered by a grid of nodes. The columns of nodes are evenly spaced in the x direction. The nodes are then evenly spaced in the y direction, bearing in mind that the flow region may have a bump, and that the y coordinate of nodes over the bump must be adjusted each time the bump parameters change. Two parameters specify the density of nodes:

NX	The number of x grid lines will be 2***NX**-1.
NY	The number of y grid lines will be 2***NY**-1.

The mesh parameter h may be determined by the computations:

$$\mathbf{HX} = \mathbf{XLngth}/(\mathbf{2} * (\mathbf{NX} - \mathbf{1})) \tag{17.9}$$

$$\mathbf{HY} = \mathbf{YLngth}/(\mathbf{2} * (\mathbf{NY} - \mathbf{1})) \tag{17.10}$$

$$\mathbf{H} = \mathbf{Min}(\mathbf{HX}, \mathbf{HY}) \tag{17.11}$$

Of course, this calculation ignores the fact that, for various bump parameter values, the y mesh spacing will differ from **HY** in the region above the bump.

17.4.3 Cost Functional Input

In our usual formulation, the cost functional is determined by picking a set of target parameters, generating the corresponding flow solution, and sampling the flow field along some specified vertical profile line. Then the integrals of the discrepancy between the components of the target flow and the current flow are computed. Certain penalty integrals may also be computed. The cost functional is then computed as a weighted sum of these integrals. The input that specifies this process includes:

ParTar	The target parameters.
WateB	Multiplier for the bump slope penalty integral.
WateP	Multiplier for p discrepancy integral.
WateU	Multiplier for u discrepancy integral.
WateV	Multiplier for v discrepancy integral.
XProf	The x coordinate of the profile line.

17.4.4 Initial Parameter Input

We generally prefer to initialize our parameters to zero. However, this choice is left up to the user. The user also specifies which parameters are allowed to vary during the optimization. Typically, for instance, the Reynolds number is held fixed.

Para1	The initial parameter values.
Iopt	Indicates which parameters may be varied.

17.4.5 Simple and Newton Iteration Input

For each new set of parameters, an estimate of the flow solution is made. The true flow solution would satisfy $F(U, V, P) = 0$, where F represents the discretized Navier Stokes equations and the boundary conditions. Assuming our flow solution estimate does not satisfy this requirement, we apply a hybrid algorithm involving Picard iteration followed by Newton iteration to try to reduce the discrepancy. The user controls this scheme through the following variables:

IJac	The Jacobian matrix will be updated every **IJac** steps.
MaxNew	The maximum number of Newton steps per iteration.
MaxSim	The maximum number of Picard iteration steps.
TolNew	The Newton convergence tolerance.
TolSim	The Picard convergence tolerance.

17.4.6 Cost Gradient Input

The cost gradient $\dfrac{\partial \mathcal{J}^h(\beta)}{\partial \beta_i}$ must be approximated for the optimization. The user controls how this is done:

IGrad	0, no gradient estimate is made
	1, use the chain rule on discretized sensitivities.
	2, use chain rule on finite coefficient differences.
	3, use chain rule on adjusted finite coefficient difference differences.
	4, use finite cost function differences.

17.4.7 Optimization Input

The user controls the optimization.

MaxStp	The maximum number of optimization steps.
TolOpt	The optimization convergence tolerance.

17.5 Algorithm Details

After reading the user input, the program must set up the data defining the cost functional. To do so, the program must produce the flow solution for the target parameters **ParTar**. It makes an initial guess of

$$\mathbf{UTar} = 0. \tag{17.12}$$

A Newton iteration then begins with this starting point, and proceeds until the convergence criterion is met. The resulting flow field is saved in its entirety for use in the cost functional.

The current estimate **ParNew** for the optimizing parameters is initialized to the user values in **Para1**. The loop expects "old" quantities evaluated at the previous solution, and we initialize them as

$$\mathbf{ParOld} \leftarrow \mathbf{ParNew} \tag{17.13}$$

$$\mathbf{UOld} \leftarrow 0 \tag{17.14}$$

$$\mathbf{dUdPOld} \leftarrow 0 \tag{17.15}$$

Inside of the optimization loop, our task is to compute the flow solution, and associated quantities, for the current set of parameters. We assume that we have the flow solution **UOld** corresponding to some previous parameter values **ParOld**, and some approximation to the sensitivities of that solution, **dUdPOld**. Our initial estimate of the new value of **U** is then:

$$\mathbf{U}(\mathbf{I}) \leftarrow \mathbf{UOld}(\mathbf{I}) + \sum_{\mathbf{J=1,NPar}} \mathbf{dUdPOld}(\mathbf{I},\mathbf{J})\ (\mathbf{ParNew}(\mathbf{J}) - \mathbf{ParOld}(\mathbf{J})). \qquad (17.16)$$

We now may take several steps of Picard iteration. On each step, we use the current estimate of the solution **U** to linearize the function **F(U)=0**, replacing it by the linear system **H(U,UNew)=0**. We solve for **UNew**, set **U** to **UNew**, and repeat.

The Newton iteration begins at the improved starting point **U** returned by Picard iteration. The Newton iteration seeks an approximate solution of the set of equations:

$$\mathbf{F}(\mathbf{U}) = 0. \qquad (17.17)$$

To do so, it requires the formation and factorization of the Jacobian matrix **FPrime(U,Para)**. A factored value of **FPrime** from a previous step may be re-used, for efficiency, as specified by the user input quantity **Jac**. The Newton iteration terminates successfully when the maximum norm of **F(U)** is less than **TolNew**. If the Newton iteration does not terminate after **MaxNew** steps, then a fatal error occurs, and the program halts.

We may now evaluate the cost functional at the new solution. To do so, we typically are required to evaluate the integral of **(U(x,y)-UTar(x,y))**2** along the line **x=XProf**. (In some cases, we may need to compute other terms as well). The integral is estimated using Gaussian quadrature, and saved as **Cost**.

It is now necessary to estimate the cost functional gradient. If the discretized sensitivities are to be used, then for each varying parameter component **J**, we set up **RHS(J)**, the right hand side of the sensitivity system, and solve the set of equations:

$$\mathbf{FPrime}\ \cdot \mathbf{dUdP} = \mathbf{RHS} \qquad (17.18)$$

for the J-th component of the sensitivities, **dUdP**.

If a finite coefficient difference approach is being used, then for each varying parameter component **J**, we temporarily set

$$\begin{aligned}\mathbf{Par2} &\leftarrow \mathbf{ParNew}, \text{ except that}\\ \mathbf{Par2}(\mathbf{J}) &\leftarrow \mathbf{ParNew}(\mathbf{J}) + \mathbf{Del}(\mathbf{J})\\ \mathbf{U2} &\leftarrow \mathbf{U}\end{aligned}$$

and carry out a Newton iteration on **U2** to get a satisfactory flow solution. Then the finite difference sensitivities are computed as

$$\mathbf{dUdP}(\mathbf{I},\mathbf{J}) \leftarrow \frac{(\mathbf{U2}(\mathbf{I}) - \mathbf{U}(\mathbf{I}))}{\mathbf{Del}(\mathbf{J})} \tag{17.19}$$

although, if shape parameters are involved, we may also want to add the adjustment term to account for movement of the nodes associated with the coefficients.

We may now use the approximated sensitivities, plus the partial derivatives of the cost functional with respect to the state variables, **dCdU**, to estimate the partial derivatives of the cost with respect to parameter **J**, using the chain rule:

$$\mathbf{dCdP}(\mathbf{J}) \leftarrow \sum_{I=1,N_w} \mathbf{dCdU}(\mathbf{I}) * \mathbf{dUdP}(\mathbf{I},\mathbf{J}) \tag{17.20}$$

Alternatively, we may approximate the partial derivatives of the cost functional directly by finite differences. In that case, one at a time, we vary a single parameter, compute the cost functional, and form the appropriate finite difference quotient that estimates the derivative.

Our cost function **Cost** and cost gradients **dCdP** are now passed to the optimization code, which makes the determination of whether the optimization loop should be continued with a new estimate of the minimizer, or terminated because of convergence or errors.

17.6 Computational Optimization

Well-written software is available to treat numerical optimization problems of the type we have considered. The program we are using incorporates a minimization package written by Gay [12], published as ACM TOMS Algorithm 611. Attractive features of this package include:

- Its status as an ACM Algorithm;
- The user can choose to provide the gradient and Hessian, or have the Hessian, or both Hessian and gradient, approximated internally. This allows us to compare the use of optimization with no derivatives, or with approximate derivatives using sensitivities or finite differences. It also means we can later try to exploit second derivative approximations if we can make them;
- It may be called in "reverse communication mode". This makes it very easy to work with functions that require extensive preprocessing before evaluation;
- The source code was publicly available through **netlib**[11];
- The program is easily portable to a variety of computers.

The program may be categorized as a *BFGS model/trust region method.*

The program requires that the user supply a starting point β_0, a tolerance ϵ and an evaluation procedure for $J(\beta)$. If the user does not supply an evaluation procedure for the gradient of J, then this will be estimated using finite differences. The program also maintains an internal estimate of the Hessian matrix $H(\beta)$.

The point β_0 is taken as the initial estimate for the minimizer. The program will proceed in a series of steps, with each step being an attempt to replace the current estimate for the minimizer by a better estimate, that is, by a point with a lower functional value.

Naturally, some times the program will find, instead, a point with a higher functional value. These points will be rejected, and the program will repeat its search based at the old point, using greater caution, to produce a new candidate. This insistence on progressive functional decrease means that the program cannot "climb up" out of a depression corresponding to a local minimum.

On each step of the iteration, the program uses recent functional values and exact or approximate information about the gradient and Hessian to construct a quadratic *model* of the functional's behavior in a small neighborhood of the current estimate. This neighborhood, in which the program believes its model to be a good approximation, is called the *trust region.* It then analyzes the model, and estimates the direction and distance of a minimizer of this model. If the estimated minimizer lies outside the trust region, the program chooses instead the point within the trust region that is closest to the estimated minimizer.

Having determined its candidate for a better minimizer, the program then requires the user to evaluate the functional at this new point. The new functional value is used to update the model. If the functional value is higher than the lowest value seen so far, then the point is not accepted, the current trust region is reduced in size, the model is revised, and a new prediction is made. Otherwise, the estimated minimizer is accepted, a new trust region and model are constructed around the new point, where the new prediction is made.

The tolerance ϵ is used in order to determine when to halt the iterative optimization, which could otherwise continue forever, assuming perfect computational accuracy. The code stops when it finds that certain things are "small" compared to ϵ. Such criteria include:

- the gradient vector of the cost functional is small;
- the distance between the current point and the minimizer is believed to be small;

- the latest steps have produced only small changes in the estimated minimizer;
- the latest steps have produced only small changes in the value of the functional.

Normally, an optimization is fairly uneventful; the optimization software might proceed slowly, or make a series of poor choices for the minimizer, but it is very rare that it is unable to proceed. One case that will cause problems, and which the optimization code will notice, occurs if the user supplies incorrect gradient information. If the gradient is quite wrong, then the optimization code will find that for points in the direction of the negative gradient, the functional actually increases rather than decreases, and that this problem persists even for very small steps. In such a case, the optimization code will return with the message "False convergence". In this case, the word "convergence" is only meant to convey the fact that no further progress can be made.

Chapter 18

SUMMARY AND CONCLUSIONS

In this study, our primary interest has been to demonstrate, in a finite element method setting, the use of discretized sensitivities of geometric parameters to approximate the cost gradient in a flow optimization.

For the problems we have studied, we have seen that the finite element form of the discretized sensitivity equations are quite simple to formulate, inexpensive to solve, and practical to use. In particular, the discretized sensitivities can be computed at almost no cost at the end of the Newton iteration that solves the current state equation. These discretized sensitivities may then be used in place of the true sensitivities of the discrete solution. However, discretized sensitivities are only an approximation to the discrete sensitivities, and in our case, we expect shape parameter sensitivities to have an approximation error that depends on the discretization size h.

We have seen that the finite coefficient differences, (if properly adjusted in the case of shape parameters), are another valuable approximation to the discrete sensitivities. An advantage of finite coefficient differences is that higher accuracy can be achieved simply by decreasing the parameter perturbation $\Delta\alpha$, rather than by decreasing the mesh parameter h. A second advantage is that finite coefficient differences are good approximations whether or not the differentiation and discretization operators commute.

In contrast, the computation of the discrete sensitivities can be a daunting task. The sensitivity equation will include not merely the expected terms with a physical interpretaion, but also terms that arise from changes in the discretization process itself, such as changes in node placement and element shape. These terms have no physical meaning, and their proper treatment requires tedious effort.

Although the discretized sensitivities were inexpensive to compute and generally sufficiently accurate for our purposes, we have seen cases where the limited approximation power caused the optimization algorithm to fail. Since our horizontal velocities u were solved with an accuracy of $O(h^2)$, the spatial derivatives have an accuracy of $O(h)$, and this in turn limits the accuracy of our discretized sensitivities to $O(h)$. The same reasoning suggests that, for our particular problem and boundary conditions, if we improved the velocity approximation to order $O(h^n)$, the shape parameter sensitivities would be accurate to order $O(h^{n-1})$.

A number of successful computations were made with the discretized sensitivities. These computations involved shape optimization at a variety of Reynolds numbers. In cases where accuracy problems did not intrude, the discretized sensitivities did indeed show themselves to be an efficient tool for making linear estimates of the solution behavior and, in optimization, for approximating cost gradients.

On the other hand, the discretized sensitivities can be poor approximants of the discrete sensitivities, even in cases where the finite element method has done a good job approximating the solution of the continuous problem. The errors in such a case can best be analyzed by also computing the finite coefficient differences, or the finite cost gradient differences for comparison.

There is a reason we have explored discretized sensitivities and finite coefficient differences as means of approximating the discrete sensitivities. It is because the discrete sensitivities contain so much useful information; they provide the cost gradient in an optimization, they show when the cost functional is well conditioned; they give physical insight into the relationship between parameters and flow solutions; they are needed when using continuation to reach higher Reynolds number solutions; and they are useful in making linear estimates of the solution or cost function for nearby parameter values.

As the Reynolds number increases, and the finite element mesh parameter h is reduced, the size of the Jacobian increases, and the cost of evaluating and factoring it predominates the cost of computing a flow solution, and hence of carrying out an optimization. Our efficiency studies suggest that the effectiveness of some simple techniques, such as holding the Jacobian fixed during a particular Newton process, or only evaluating the Jacobian when the parameters have changed by more than some given tolerance.

A number of questions remain open as fruitful areas of further research:

- First, further study of the behavior of the discretized sensitivities at higher Reynolds numbers is needed, but computing limitations restricted our investigations to values no higher than $Re = 1000$.
- Secondly, it would be useful to have a carefully worked out estimate of the approximation error between the discrete sensitivities and the discretized sensitivities. In this connection, it is vital to examine the errors in the boundary condition for the discretized sensitivities, to quantify the influence these errors have on the computed discretized sensitivities, and to produce estimates of the behavior of this error as the mesh parameter h goes to zero.
- Finally, practical algorithmic procedures are needed for estimating the approximation error in the discretized sensitivities and reducing it when necessary, perhaps by local

refinement of the grid or the local use of higher order elements.

Bibliography

[1] J Boland and R Nicolaides, *The Stability of Finite Elements Under Divergence Constraints*, **SIAM Journal on Numerical Analysis**, Volume 20, Number 4, pages 722-731, August 1983.

[2] Jeff Borggaard, **The Sensitivity Equation Method for Optimal Design**, PhD Thesis, Virginia Polytechnic Institute and State University, Department of Mathematics, 1994.

[3] Jeff Borggaard, John Burns, Gene Cliff and Max Gunzburger, *Sensitivity Calculations for a 2D, Inviscid Supersonic Forebody Problem*, in **Identification and Control of Systems Governed by Partial Differential Equations**, H T Banks, R Fabiano, K Ito, editors, SIAM Publications, pages 14-24, 1993.

[4] John Burkardt and Janet Peterson, *Control of Steady Incompressible 2D Channel Flow*, **Flow Control**, The IMA Volumes in Mathematics and Its Applications, Volume 68, Springer Verlag, New York, 1995.

[5] John Burkardt and Janet Peterson, *The Role of the Cost Functional in a Shape Optimization*, **Proceedings of the Fourth Bozeman Conference on Computation and Control**, Birkhaeuser, to appear 1995.

[6] John Burns, Eugene Cliff, Max Gunzburger, *Equations for the Sensitivities for the 2D, Inviscid, Supersonic Forebody Problem*, ICAM Report 91-07-05, Interdisciplinary Center for Applied Mathematics, Virginia Polytechnic Institute and State University, Blacksburg, VA, July 1991.

[7] Hayri Cabuk and Vijay Modi, *Optimum Plane Diffusers in Laminar Flow*, **Journal of Fluid Mechanics**, Volume 237, pages 373-393, April 1992.

[8] R G Carter, *On the Global Convergence of Trust Region Algorithms Using Inexact Gradient Information*, **SIAM Journal on Numerical Analysis**, Volume 28, Number 1, pages 251-265, February 1991.

[9] Carl DeBoor, **A Practical Guide to Splines,** Springer Verlag, New York, 1978.

[10] J E Dennis and Robert Schnabel, **Numerical Methods for Unconstrained Optimization and Nonlinear Equations**, Prentice Hall, Englewood Cliffs, 1983.

[11] Jack Dongarra and Eric Grosse, *Distribution of Mathematical Software Via Electronic Mail*, **Communications of the ACM**, Volume 30, pages 403-407, 1987.

[12] David Gay, *Algorithm 611, Subroutines for Unconstrained Minimization Using a Model/Trust Region Approach,* **ACM Transactions on Mathematical Software**, Volume 9, Number 4, pages 503-524, December 1983.

[13] Vivette Girault and Pierre-Arnaud Raviart, **Finite Element Methods for Navier-Stokes Equations**, Springer Verlag, Berlin, 1986.

[14] Ph Guillaume and M Masmoudi, *Computation of High Order Derivatives in Optimal Shape Design*, **Numerische Mathematik**, Volume 67, pages 231-250, 1994.

[15] Max Gunzburger, **Finite Element Methods for Viscous Incompressible Flows**, Academic Press, San Diego, 1989.

[16] Max Gunzburger and Janet Peterson, *On Conforming Finite Element Methods for the Inhomogeneous Stationary Navier-Stokes Equations*, **Numerische Mathematik**, Volume 42, pages 173-194, 1983.

[17] J Haslinger and P Neittaanmäki, **Finite Element Approximation for Optimal Shape Design**, John Wiley and Sons, Chichester, 1988.

[18] D Huddleston, *Development of a Free-Jet Forebody Simulator Design Optimization Method*, AEDC-TR-90-22, Arnold Engineering Development Center, Arnold AFB, TN, December 1990.

[19] Ohannes Karakashian, *On a Galerkin-Lagrange Multiplier Method for the Stationary Navier-Stokes Equations*, **SIAM Journal on Numerical Analysis**, Volume 19, Number 5, pages 909-923, October 1982.

[20] Hongchul Kim, **Analysis and Finite Element Approximation of an Optimal Shape Control Problem for the Steady-State Navier-Stokes Equations**, PhD Thesis, Virginia Polytechnic Institute and State University, Department of Mathematics, 1993.

[21] O A Ladyzhenskaya, **The Mathematical Theory of Viscous Incompressible Flow**, Second Edition, Gordon and Breach, New York, 1969.

[22] L Landau and E Lifshitz, **Fluid Mechanics**, Pergamon Press, Oxford, 1987.

[23] Werner Rheinboldt and John Burkardt, *A Locally Parameterized Continuation Process*, **ACM Transactions on Mathematical Software**, Volume 9, Number 2, pages 215-235, July 1983.

[24] Werner Rheinboldt and James Ortega, **Iterative Solution of Nonlinear Equations in Several Variables**, Academic Press, New York, 1970.

[25] Roger Temam, **Navier-Stokes Equations**, Revised Edition, North-Holland, Amsterdam, 1979.

VITA

The author's improbable career may bring hope to those still struggling to find their proper place in life.

He went to MIT with a vague interest in Mathematics, only to be blighted by a course in number theory. An attempt at a physics major came to an end in the lab, when he managed to make a vacuum tube with two anodes. He started a computer course, but couldn't get the hang of it. Almost every term, though, he took courses in German and in writing. It was only by the purest luck that he was able to graduate, as the first recipient of a degree in the newly blessed field of "Science and Humanities".

Graduated, jobless, planless, he decided to take some summer classes at the University of Pittsburgh. He enjoyed his course in sophomore calculus so much that, at the end, he practically begged his instructor, "Is there some way I could do more of this?" The instructor arranged for him to meet Professor Jim Fink, who got him provisional admission, conditional on surviving a term of catchup work. That first term involved back-to-back classes from 9 AM to 3 PM, with no lunch break, but was the first time he had enjoyed school in years. He was so excited he wrote to two old MIT classmates who were currently returning cars for Avis, and they both also enrolled in graduate school, also in Mathematics.

After two years of abstract mathematics, including topology, set theory and logic, he felt unprepared to take the preliminary exams and simply didn't show up, forfeiting all further departmental support. It was only by a miracle that he found himself saved, working on a one year grant with Professor Andras Szeri of the Mechanical Engineering Department, who required him to learn FORTRAN, the use of a timesharing computer, and numerical analysis.

The course in numerical analysis, taught by Professors Charles Hall and Tom Porsching, is fixed in his mind for all time. He had enjoyed the preliminary material, but expected things were about to go downhill when ordinary differential equations arrived. But then the professor sketched the field of derivatives and said "Suppose we're at point X, where the solution is equal to Y, and we want to estimate the value of the solution at point X + Δ X. What's a simple estimate we can make, using this picture?" This was the moment when he fell in love with numerical analysis.

Gradually, he drifted from graduate work into full time computer support for Hall and Porsching's research group in the Mathematics department. He worked on a variety of fluid codes for the Air Force and EPRI, and a sheet metal forming program for GM. He was lucky enough to work with Professor Werner Rheinboldt in developing a version of his continuation program suitable for submission as an ACM algorithm. He also spent a great deal of time writing educational software for Professor Charles Cullen, from whom he learned a lot about proper computer educational techniques.

He then took a job at the newly opened Pittsburgh Supercomputing Center, as a "mathematical software librarian", which initially meant supporting the IMSL and NAG libraries. Gradually this job came to include installing, testing, and documenting software, porting software to a variety of Cray computers, debugging and optimizing user programs, team-teaching workshops for users, both on site and at a number of universities, and working on an outreach program that brought the joys and headaches of computing to students and teachers in local high schools.

However, after five years, he was ready for something new. Professor Janet Peterson, who had known him at the University of Pittsburgh and was now at Virginia Tech, had urged him several times to return to graduate school and finish his degree. Now she suggested he come down to Virginia Tech and do so. Professor Max Gunzburger also applied some gentle charm, offering to supervise his work, and introducing him to Professors John Burns, Gene Cliff, and Terry Herdman, the likely members of his defense committee.

It seemed like the last chance of a lifetime, and so he quit his job, sold his house, rode down to Blacksburg with six boxes of belongings, and hasn't looked back.