SELECTED OPTIMIZATION PROCEDURES FOR CFD-BASED SHAPE DESIGN INVOLVING SHOCK WAVES OR COMPUTATIONAL NOISE

By

Robert P. Narducci

A DISSERTATION SUBMITTED TO THE FACULTY OF
VIRGINIA POLYTECHNIC INSTITUTE AND STATE UNIVERSITY
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY
IN
AEROSPACE ENGINEERING

B. Grossman, Chairman

R.T. Haftka

R.W. Walters

W.H. Mason

E. Cliff

May 1995

Blacksburg, Virginia

c.2

SELECTED OPTIMIZATION PROCEDURES FOR CFD-BASED SHAPE DESIGN INVOLVING SHOCK WAVES OR COMPUTATIONAL NOISE

by
Robert P. Narducci

Bernard Grossman, Chairman
Department of Aerospace and Ocean Engineering

(ABSTRACT)

This work addresses many problems associated with designing aerodynamic shapes using computational fluid dynamics (CFD) codes. The investigation focuses in the transonic flow regime where shock waves may have an adverse effect on the convergence of the optimization process. In particular, the interaction of the flow discontinuity with the discrete representation of the design problem may cause the objective function to be non-smooth. Methods for robust optimization of the non-smooth functions are presented.

The dissertation is divided into two parts. The first part investigates a simple model problem involving quasi-one-dimensional flow in a duct. The flow field computation is simple and contains many of the elements present in more complicated fluid flow problems. The optimization involves finding the cross-sectional area distribution of a duct that produces velocities which closely match a targeted velocity distribution containing a shock wave. The objective function which quantifies the difference between the targeted and calculated velocity distribution becomes non-smooth due to the presence of the shock in the discretized field. Two techniques for derivative-based optimization are offered to resolve the difficulties associated with the non-smoothness of the objective function. The first technique, shock-fitting, involves careful integration of the objective function through the shock wave. The second technique, coordinate straining with shock penalty, uses a coordinate transformation to align the calculated shock with the target and then adds a penalty

proportional to the square of the distance between shocks. These techniques are evaluated and tested using several methods to compute the derivatives, including finite-differences, direct and adjoint methods.

The above two techniques rely on accurate estimations of the shock position, which may not be available for the general case. In the second part of the dissertation, we present an optimization method to solve the difficult model design problem requiring no information about the shock. The optimization begins with the construction of a response surface that smoothly approximates the objective function. Here the response surface is a least squares polynomial fit to carefully selected design points. By minimizing the response surface we can obtain a first guess for a reasonable design. Optimization may continue in one of two ways. In the first method, we probe a small region of the design space around the minimum and perform another response surface minimization. In the second method we switch to a derivative-based method assuming that in the small region around the minimum the function is smooth. In addition to the one-dimensional duct problem, two other shape design problems involving two-dimensional flow are solved to demonstrate the efficacy and robustness of the response surface method. One involves the inverse design of a bump in a transonic channel flow. The other involves the design of an airfoil for transonic flight.

ACKNOWLEDGEMENTS

I would like to thank my advisor and committee chairman, Dr. Bernard Grossman, for his guidance and his timely reading of this dissertation. Also, many thanks to the other committee members. Specifically, to Dr. Raphael Haftka for sharing his deep understanding of optimization, to Dr. Robert Walters for his assistance in developing the CFD code ErICA, to Dr. Eugene Cliff for all his advice on the diffuser design problem, and to Dr. William Mason for sharing his experiences with computational aerodynamics results. In addition I would like to thank Dr. Layne Watson and Susan Burgee for their suggestions in reference to the parallel computer work.

I am grateful to M. Salas of NASA Langley for supervising and funding this work through NASA grant NAG 1-1466.

Special thanks to my father and mother who inspired me to grow in knowledge and encouraged me along the way; to my wife for her patience and love; and to my friends in the Sunroom (without whom I would have graduated two months earlier).

Thank you Lord for teaching me to strive for perfection.

TABLE OF CONTENTS

PART TWO
RESPONSE SURFACE METHOD

LIST OF TABLES

LIST OF FIGURES

INTRODUCTION

Shape optimization problems in aerodynamics have recently captured the interest of many researchers as solutions to fluid dynamics problems have become less computationally restrictive. The increase in computational efficiency can be attributed to the incorporation of new procedures such as multigrid[1] and preconditioning,[2] but also to computer improvements, such as the development of parallel machines. These advances have lead to many new methods to solve the design problem as flow solutions for many different configurations are available to the designer. Our work here focuses on the modification of existing design procedures and the implementation of a response surface method applied to transonic flow design problems involving shock waves.

Design problems can be categorized into two main types, *inverse* and *direct*. Of the inverse type, we speak of *flow field design* and *surface flow design*. The flow field design problem is formulated by specifying some feature throughout the flow, *e.g.* a shock-free flow. The solution describes the shape of the solid boundary such that the flow field satisfies the condition specified. The best known method for flow field design is the hodograph method applied to two-dimensional, transonic, shock-free airfoils.[3–11] This method transforms the partial differential equations governing potential flow to the hodograph plane, where they appear as linear equations. Then, by superposition of simple solutions, complex, shockless flows are constructed. Although some excellent airfoils have been designed by this method, some "optimized" shapes have been known to have open or fish-tail trailing edges.

Another flow field design method for transonic, shock-free airfoils is the fictitious gas method invented by Sobieszczky *et al.*.[12] This method uses a false density-Mach number relation in regions where the flow would be supersonic. The fictitious

relation is defined such that the flow is subsonic in this region, eliminating the possibility for the existence of shocks. Along the sonic line the governing equations for both regions (the isentropic region and the fictitious gas domain) are satisfied. By reverting to the normal isentropic relation in the fictitious gas region, the new airfoil shape is determined by forcing the stream function to have a constant value everywhere along the airfoil surface. The fictitious gas method is easier to implement than the hodograph method and does not suffer from designs with open or fish-tail trailing edges. Applications of this method can be found in references 13-17.

Surface flow design is the more common approach to inverse design and more work has been done with this problem than the flow field design type. In surface flow design, some aerodynamic quantity is specified along a boundary, *e.g.* pressure on an airfoil surface. The solution to the problem describes the shape of the boundary that will generate the specified distribution. The list of literature involving shape design via surface flow specification in very long. Progress in this field is well summarized by Holst *et al.*,[18] and more recently by Dulikravich.[19]

Surface flow inverse design applied to airfoils was first formulated by Lighthill.[20] Lighthill pointed out that the inverse problem in airfoil design is well-posed provided the target velocity (or pressure) distribution satisfies three constraints. Two constraints are associated with the airfoil's trailing edge gap. The third constraint requires that the target velocity distribution is compatible with the specified free stream velocity. The problem is well-posed provided the target distribution is formulated in terms of parameters which guarantee the constraints are satisfied.

In incompressible flow problems, an explicit relationship for the parameters exists so that the target distribution satisfies the constraints.[21–23] For compressible flow, these explicit relationships have only been found for Karman-Tsien type gas.[24] No explicit relationship has been found for compressible perfect gas. Tranen,[25]

Carlson,[26] and Shankar[27] have been able to design airfoils by satisfying two of the three constraints. They were not able to satisfy the target distribution to within arbitrarily small tolerances since the free stream condition was not satisfied. Volpe and Melnik[28] were able to satisfy all three constraints for a compressible perfect gas problem by numerically determining the values of the parameters in the target distribution as part of the problem.

In direct design problems some objective is quantified as a function of one or more design variables. In airfoil design typically the drag, formulated as a function of the airfoil shape, acts as the objective function. The objective is met when the function is minimized (or maximized). The solution to the direct problem is the shape described by the set of design variables which minimizes (or maximizes) the objective function. One key advantage to solving the direct problem over the inverse problem is that the designer does not have to rely on experience to determine the aerodynamic target distribution. Also, many times the data specified for the inverse problem may not correspond to a feasible design. Progress in this field is extensive and is well summarized in references 18 and 19. A brief overview of some optimization methods available for solving inverse and direct problems are presented in the next few paragraphs.

Methods for optimization of surface flow inverse types and direct design types can be classified according to how they use function, gradient, and Hessian information. Algorithms which use only function values are classified as zeroth-order methods. First-order methods use function and gradient information and second-order methods include the use of the Hessian.

There are many zeroth-order methods. Of rising popularity is the genetic algorithm (GA). The GA is a search procedure whereby designs are completely represented by a string of genes. Instead of starting from a single design point,

the GA uses a population of designs that are compared and ranked in relation to one another. Using a simple breeding method, two genetic strings from the population are recombined to produce a new design of (hopefully) higher merit. Breeding continues until the population is replenished. The best design from the previous population is added to the population. The algorithm continues over many generations, always keeping the best design in the current population set. The GA works because the probability of a genetic string being selected for breeding is related to its rank in the population. In this way designs of higher quality are selected more often for breeding. Also a random chance for genetic mutation is usually incorporated to allow for genes not found in the population to enter into the design process. GA's have been used in optimization since the 1980's,[29] and the extension to shape design has been made more recently *e.g.* references 31-33. A drawback of the GA is the large amount of function evaluations required to complete an optimization. This can make a design prohibitively expensive if CFD solutions are needed to evaluate the objective function.

Another zeroth-order method involves probing the design space at locations where the optimum is statistically most likely to occur. GROPE (Global R^d Optimization when Probes are Expensive), invented by Elder,[34] is an extension to multiple dimensions of the univariate search method introduced by Kushner.[35] It is an ideal algorithm for multipeak functions in low dimensions as it attempts to balance the competing aims of sampling in the vicinity of a known peak and exploring new regions. One attractive feature of this algorithm is the apparent low number of samples required to locate the minimum. However, implementation to higher dimensions is complex. To the author's knowledge GROPE has not been applied to shape optimization problems.

The response surface or simulation approach is another way to optimize functions when derivatives are inaccurate or unavailable. Here sequential approxima-

tions to the function and constraints are made by sampling a portion of the design space. With the acquired data, curves are constructed, usually by a least-squares approach, to simulate the objective function and it's constraints. Minimization will lead to an improved design if the curves model the function and constraints accurately. The size and shape of subsequent subregions can be changed based on the results of the minimization. This method is particularly useful when the function is noisy as it does not require costly or noise-contaminated derivatives. This approach originated from the field of design of experiments and experimental optimization where the results and conclusions that can be made rely heavily on the manner in which data were collected.[36] The application of response surfaces to numerical experiments is straight forward as noisy numerics creates the same circumstances as experimental tests. The approach has been used successfully in a multitude of noisy structural optimization problems.[37–44] The extension to other disciplines is developed in references 45 and 46.

Much of the design work today uses derivative-based methods. Many of the optimization methods are well established and are covered in text books on the subject *e.g.* reference 47. In aerodynamic shape design, work has focused on efficiently and accurately computing the first derivatives, either analytically or by numerical means. Due to the excessive cost involved in shape design, the Hessian is often approximated, for example by BFGS updates, when second order methods are utilized. Recent advances in first-order optimization involving transonic flow are presented in references 48-61.

It was discovered in our earlier work,[48] that the shock wave, if ignored as a discontinuity, can slow convergence or cause the optimization to fail. In CFD solutions the difficulty in representing a discontinuity on a grid can cause noise in the evaluation of design derivatives and can cause waviness or discontinuities in the

objective function. An important conclusion in reference 48 is that as the quality of the flow solution improves and the shock wave resolution sharpens, the more the shock wave tends to disrupt the optimization process. Advances in the quality and efficiency of CFD codes have allowed the engineer the luxury of designing shapes using flow solutions with very sharp shocks. However methods must be developed to counter the adverse effects that sharp shocks have on the optimization process. This establishes the motivation for this work.

The motivation began with our initial failure to reproduce the results presented by Frank and Shubin.[49,50] In their work, they compared several methods for computing design derivatives as applied to an inverse design of a duct involving quasi-one-dimensional, transonic flow. The comparison was based on the accuracy and efficiency of computing the design derivatives. The study included finite-differences, direct, adjoint, and all-at-once methods. Some of these methods require differentiation or integration over sharp discontinuities in the objective function or governing equations, yet no special provision was made for the existence of a discontinuity. Despite this, Frank and Shubin were reasonably successful in their optimization. They were able to improve their design because the shock placement of the initial design was very near to the target's and thus the optimization did not suffer many ill-effects from the non-smooth objective function. Any attempt to reproduce their converged design from different initial design points will end in failure because the discontinuities are not properly accounted for.[48]

Others have used slight variations of the design problem of Frank and Shubin to propose techniques to avoid convergence difficulties associated with the shock wave.[51–55] One method, proposed by Iollo *et al.*,[51] is the shock-fitting technique which involves placing a break point at the shock position. Integration and differentiation are carefully performed around the break point. In this way they were

able to successfully recover the target solution to within double precision machine accuracy using an adjoint method. Their initial conditions had the shock wave significantly far from the target.

Shenoy and Cliff[52] used the shock-fitting technique in an optimum control approach to solve Frank and Shubin's inverse problem. A variation of this work by Wu, Cliff, and Gunzburger[53] applied the optimal control problem to a two-dimensional version of the inverse design problem. They used the shock-fitting technique in the formulation of the control problem, but smoothed the shock in the flow solution before the evaluation of the objective function. This gave them a well-behaved objective function.

Borggaard *et al.*[54] successfully solved Frank and Shubin's problem from arbitrary initial conditions by using a smoothing function when computing design derivatives. The objective function was evaluated using the solution with the sharp shock. Borggaard and Burns extended this technique to a two-dimensional case.

Reuther and Jameson,[52–60] dealt with the shock wave indirectly in their work on transonic airfoil design. They developed an adjoint approach based on control theory for computing the design derivatives. However in their calculations they used flow solvers that sufficiently smeared the shock wave. Here, it is enough to smear the discontinuity over 4 or 5 grid points. Thus in their formulations, no special treatment of the shock wave was necessary. To avoid waviness in the objective function resulting from the shock wave, they applied a smoothing transformation to the objective function. In this way, differentiation of the objective function did not involve differentiation over sharp discontinuities.

Gilmore and Kelley[62] have recently developed a derivative-based optimization procedure for dealing with noisy functions with many local minima. This algorithm is particularly applicable to objective functions which can be expressed as a sum of

a simple function, such as a convex quadratic, and a high frequency, low amplitude function. The method works best when the amplitude of the high frequency function decays near the minimum of the simple function. Optimization proceeds using a finite-difference gradient-based method with the step size of the finite-difference derivatives chosen to "step over" the high frequency noise. The gradient-based optimization method is repeated several times with each subsequent application using decreasing step sizes in the finite-difference calculations. This algorithm has been applied to the design of microwave devices in references 63, and 64.

The objective of this work is to present several optimization strategies for the design of shapes submerged in flows containing discontinuities. The obvious application is to transonic airfoil design. However, it is my hope that some of the strategies discussed herein will be applied within the framework of multidisciplinary optimization. This dissertation is divided into two parts.

Part I is an overview of some existing methods to compute design derivatives for derivative-based optimization methods. In particular, we investigate the adjoint and direct methods for computing design derivatives and compare the calculations to a finite-difference method. Modifications have been made to these methods to handle the flow discontinuity. In chapter one of this section, an example inverse design problem is introduced to validate the modifications. The problem involves the one-dimensional transonic Euler flow through a duct of varying cross-sectional area originally studied by Frank and Shubin.[49,50] Chapter two presents the method of optimization chosen in the design process. In chapter three, the direct and adjoint methods are applied to the example problem using a continuous approach. In the continuous approach, the derivative methods are applied to the objective function and governing equations *before* they are discretized. To properly account for the shock wave, the shock-fitting technique by Iollo is utilized. In chapter

four, the direct and adjoint methods are applied to the objective function and governing equations *after* they are put in their discrete form. This is called the discrete approach. The flow field is modified by a coordinate transformation so that the shock wave is aligned with the target. Optimization is performed on the objective function based on the transformed solution and a penalty related to some transformation parameters. Chapter five presents design results and a comparison of the methods to compute the derivatives. Part I ends in chapter six with some important conclusions.

In part II, a response surface optimization procedure for shape design in the presence of noise is developed. In this method, a response surface is fitted to the objective function within some region of the design space. A minimization of the response surface leads to a design with optimal characteristics. In chapter seven, an algorithm for multiple response surface cycles is proposed which may lead to further improvements in the design. In chapter eight of this section, a recipe for construction of the surface is given. This includes selection of the response surface shape, selection of the number of points to use in its construction, where to probe the design space to achieve the most reliable curve, and a least-squares procedure to construct the curve. Selection of the location of the points in the construction of the response surface requires the solution to an optimization subproblem. This is the D-optimal problem. We solve for D-optimality via a GA which is described in chapter nine. The response surface method is applied to the inverse design problem described in part I, but also to two-dimensional problems. The flow fields for the two-dimensional problems are solved with a self-developed Euler solver, ErICA (EuleR Inviscid Code for Aerodynamics). A user's guide for this code is presented in Appendix C. The details of these design problems are described in chapter ten. The results of the optimization and a discussion are presented in

chapter eleven. The dissertation ends with important conclusions drawn from the results presented and suggestions for future work are made.

PART
ONE

DERIVATIVE CALCULATIONS METHODS FOR FIRST-ORDER OPTIMIZATION

"Wisdom is supreme; therefore get wisdom.

Though it cost all you have,

get understanding."

—Proverbs 4:7

CHAPTER
ONE

A TRANSONIC INVERSE DESIGN PROBLEM

Before investigating methods which may be useful for calculating the derivatives in shape optimization, we first introduce a seemingly simple design problem which will allow us to examine difficulties which may arise for problems involving flows with steep gradients. The problem chosen is such that flow solutions are very cheap, yet it contains many of the elements present in more complex design problems. In particular, the problem contains a shock wave. The simplicity of the flow field allows us to resolve the shock precisely. Thus, despite the ease of solving the forward problem, the design problem is perhaps more difficult to solve than practical design problems with more complex flow solutions.

The problem involves matching a velocity distribution through a duct by controlling the cross-sectional area. The velocity is computed using quasi-one-dimensional flow theory. The details of the problem are discussed in the remainder of this chapter. We begin with a discussion of the governing equations. An analytic solution is available and is discussed next. Usually in shape optimization, we do not have the luxury of working with exact solutions. For this reason, two numerical solutions are presented. One, the Godunov solution, is an upwind solution capable of capturing the shock wave to within two grid spacings. The other uses artificial viscosity which can smear the shock considerably. Next a formulation of the design problem is presented. This subsection includes the definition of the design variables and several objective functions used in the optimization.

1.1 Governing Equations

In one-dimensional flow we consider a streamtube where the flow variables are allowed to vary in only one direction. As a consequence the cross-sectional area of the streamtube must be constant. However for streamtubes where the gradient of the area changes slowly, it is possible to neglect the three dimensionality of the flow and consider only the variation of the flow properties in the direction of the axis of the streamtube. Such a situation is considered as quasi-one-dimensional and an example is shown in figure 1.1.

The governing equations for steady quasi-one-dimensional flow are the Euler equations expressed in differential form as

$$\left\{ \begin{array}{c} \rho u A \\ (\rho u^2 + p)A \\ (\rho e_o + p)uA \end{array} \right\}_x + \left\{ \begin{array}{c} 0 \\ -pA_x \\ 0 \end{array} \right\} = 0. \tag{1.1}$$

The independent variable, x, varies from 0 to l, the length of the duct. The density is ρ, u is the velocity, A is the area, p is the pressure, and e_o is the total energy per unit mass. The equation of state for a perfect gas closes the system

$$p = (\gamma - 1)\rho e, \tag{1.2}$$

where e is the internal energy per unit mass, $e = e_o - u^2/2$, and γ is the ratio of specific heat. In our computations, γ is taken to be a constant value equal to 1.4. Frank and Shubin[49] manipulated equations (1.1) and (1.2) to get a single ordinary differential equation in the variable u

$$\frac{df}{dx} + g = 0, \tag{1.3}$$

where

$$f(u) = u + \frac{(\gamma - 1)}{(\gamma + 1)} \frac{2h_o}{u}, \tag{1.4}$$

and

$$g(u, A) = \frac{A_x}{A}\frac{(\gamma - 1)}{(\gamma + 1)}\left(u - \frac{2h_o}{u}\right). \tag{1.5}$$

Equation (1.3) with definitions (1.4) and (1.5) have been normalized. The velocity has been normalized to the speed of sound at the inlet, the total enthalpy, h_o, has been normalized to the square of the speed of sound, the area has been normalized to the square of the length of the duct, and the independent variable has been normalized to the length of the duct.

For the case where a shock exists in the duct, the characteristics of (1.3) are such that boundary values must be specified at $x = 0$ and $x = 1$. This corresponds to the entrance and exit of the duct. Following the details of the problem set forth by Frank and Shubin, we specify

$$u(0) = 1.299, \tag{1.6}$$

and

$$u(1) = 0.506. \tag{1.7}$$

There are a variety of ways to solve (1.3) with boundary conditions (1.6) and (1.7). An analytic solution exists, the derivation of which is presented in the next subsection. For the purpose of simulating more complex fluid problems for which only numerical solutions are available, we also solve the quasi-one-dimensional problem via two numerical techniques.

1.2 Exact Solution

An exact solution to (1.3) can be derived by first substituting the derivative of (1.4) along with (1.5) into (1.3). This yields

$$u_x\left(1 - \frac{\gamma - 1}{\gamma + 1}\frac{2h_o}{u^2}\right) + \frac{A_x}{A}\frac{\gamma - 1}{\gamma + 1}u\left(1 - \frac{2h_o}{u^2}\right) = 0, \tag{1.8}$$

where the subscript x refers to differentiation with respect to the independent variable. Multiplying (1.8) by the factor $-Au^2(\gamma+1)/(\gamma-1)$ gives

$$-\frac{Au_x u^2(\gamma+1)}{\gamma-1} + 2Au_x h_o + A_x u\left(2h_o - u^2\right) = 0. \tag{1.9}$$

After some algebraic manipulation we can write (1.9) as

$$-\frac{2Au_x u^2}{\gamma-1} + \left(Au_x + A_x u\right)\left(2h_o - u^2\right) = 0. \tag{1.10}$$

Using the chain rule of differentiation, the second term in (1.10) can be expressed as $(Au)_x(2h_o - u^2)$. Defining $r \equiv 1/(\gamma-1)$ and multiplying by $(2h_o - u^2)^{(r-1)}$ we have

$$Aur(2h_o - u^2)^{r-1}(-2uu_x) + (Au)_x(2h_o - u^2)^r = 0. \tag{1.11}$$

The first term in (1.11) is nothing more than $Au(2h_o - u^2)^r_x$. Again using the chain rule we arrive at the perfect differential

$$\left[Au(2h_o - u^2)^r\right]_x = 0. \tag{1.12}$$

Simple integration through continuous regions of the domain yields

$$\begin{aligned} Au(2h_o - u^2)^r &= k_1, \qquad 0 \le x \le x_s \\ Au(2h_o - u^2)^r &= k_2, \qquad x_s \le x \le 1 \end{aligned} \tag{1.13}$$

where k_1 and k_2 are the constants of integration determined by satisfying the boundary conditions (1.6) and (1.7). The shock position, x_s, is determined by satisfying the Rankine Hugoniot relation, which when written in our non-dimensional variables is

$$u^+u^- = 2h_o\frac{(\gamma-1)}{(\gamma+1)}, \tag{1.14}$$

where u^+ and u^- are the left and right values of velocity on each side of the shock wave. This solution was taken from reference 49. Obtaining the exact solution is shown graphically in figure 1.2.

1.3 Finite-volume Solutions

Numerical solutions to (1.3) may be obtained by adding a nonphysical time derivative and marching to a steady state. Consider a conservation law for a one spatial dimensional problem of the form

$$\frac{\partial u}{\partial t} + \frac{\partial f}{\partial x} + g = 0, \tag{1.15}$$

where u can be interpreted as the velocity averaged over a cell volume, the flux derivative $\partial f/\partial x$ may be interpreted as the net cell area averaged flux through the cell surface, and g interpreted as a volume averaged source term. The semi-discrete representation of (1.15) is

$$\frac{\partial u_j}{\partial t} + \frac{f_{j+1/2} - f_{j-1/2}}{\Delta x} + g_j = 0, \tag{1.16}$$

where the fluxes $f_{j\pm1/2}$ are evaluated in one of two ways. The first method is called the Godunov scheme, as implemented by Frank and Shubin,[49,50] and uses the upwind formulation according to

$$f_{j+1/2} = \begin{cases} f_{j+1} & u_j, u_{j+1} < u_* \\ f_j & u_j, u_{j+1} > u_* \\ f_* & u_j < u_* < u_{j+1} \\ \max(f_j, f_{j+1}) & u_j > u_* > u_{j+1} \end{cases} \tag{1.17}$$

where $f_{j+1} = f(u_{j+1})$, etc., and $*$ indicates sonic flow. An alternative to the Godunov method is the artificial viscosity method where

$$f_{j+1/2} = \frac{1}{2}\left[f_{j+1} + f_j - \alpha(u_{j+1} - u_j)\right]. \tag{1.18}$$

In (1.18), α is a parameter related to the artificial viscosity.

Time marching is performed using the four-stage Runge-Kutta scheme presented below[66]

$$\begin{aligned} u^{(0)} &= u^n, \\ u^{(1)} &= u^n - \frac{\Delta t}{4} R(u^{(0)}), \\ u^{(2)} &= u^n - \frac{\Delta t}{3} R(u^{(1)}), \\ u^{(3)} &= u^n - \frac{\Delta t}{2} R(u^{(2)}), \\ u^{(4)} &= u^n - \Delta t R(u^{(3)}), \\ u^{n+1} &= u^{(4)}, \end{aligned} \tag{1.19}$$

where

$$R(u) = \frac{f_{j+1/2} - f_{j-1/2}}{\Delta x} + g_j. \tag{1.20}$$

A solution using the Godunov scheme is shown in figure 1.3. In figure 1.4 several artificial viscosity solutions with various values of α are shown. Solutions with sharp shocks can be obtained using the Godunov scheme or the artificial viscosity scheme with low values of α.

1.4 Formulation of the Design Problem

The design problem presented in this section will roughly describe the problem introduced by Frank and Shubin in reference 49, and studied by others[50–55]. The objective of the problem is to recover the shape of the duct that will yield a given velocity distribution. To describe the shape of the duct, we use a set of design variables, ξ. To insure that the target can be recovered to within machine accuracy, we impose the following requirement: the target velocity distribution must be a solution to the governing equation for a feasible area distribution. The requirement is only needed to aid in the evaluation of our methods and generally is not a requirement of the methods.

To satisfy the requirement, the target velocity distribution is obtained by solving the governing equation (1.3) with the boundary condition (1.6) and (1.7) using an area distribution described by

$$A(x) = -1.390x^3 + 2.085x^2 + 1.050. \tag{1.21}$$

The distribution has the property that $A(0) = 1.05$, $A(1) = 1.745$, and the slopes are zero at the ends. The governing equation is solved to get the target velocity distribution using either the exact or the numerical methods, depending on which flow solver is used during the optimization. For example, if the optimization uses the Godunov scheme to obtain solutions to the forward problem, then the target solution is obtained using (1.21) and the Godunov scheme.

There are many ways to specify an area distribution. In this work, a cubic spline is fitted through control points along the duct. The design variables are the ordinate of the control points and physically represent the cross-sectional area of the duct at the axial location of the control points. The design cases presented in this section vary in number of design variables. In each case, the control points are evenly spaced along the axis of the duct. The inlet and exit area of the duct are fixed at the normalized values of 1.05 and 1.745 and clamped boundary conditions, *i.e.* zero slope at the ends are imposed to determine the spline uniquely. A schematic for the general case with n design variables is shown in figure 1.5. With this formulation the target area, (1.21), is an element of the feasible area distributions.

To quantify how well a calculated velocity distribution compares to the target we define an objective function of the form

$$I(\xi) = \frac{1}{2}\int_0^1 (\hat{u} - u)^2 dx, \tag{1.22}$$

where $\hat{u} = \hat{u}(x)$ is the target velocity distribution through the duct, and $u = u(x;\xi)$ is the calculated velocity distribution at the current values of the design variables.

In the discretization of the problem, the integral is approximated over a uniform mesh using the trapezoidal rule

$$I(\xi) = \frac{1}{2}\left\{\frac{1}{2}\left[(\hat{u}-u)_1^2 + (\hat{u}-u)_N^2\right] + \sum_{i=2}^{N-1}(\hat{u}-u)_i^2\right\}\Delta x, \tag{1.23}$$

where N is the number of grid points. Boundary conditions specify u at the inlet and exit to match the target reducing (1.23) to

$$I(\xi) = \frac{1}{2}\sum_{i=2}^{N-1}(\hat{u}-u)_i^2\Delta x. \tag{1.24}$$

An optimum design is reached when (1.24) is a minimum. When the requirement mentioned earlier is satisfied the optimal design is reached when (1.24) is equal to zero to machine accuracy.

Making the trapezoidal rule approximation without regard to the shock wave in the velocity distribution can result in a non-smooth or discontinuous objective function. Consider a univariate design case with the design variable located at $x = 0.5$. The objective function evaluated with the exact solution for both u and $\hat{u}$ and plotted over a range from $\xi = 1.1$ to $\xi = 1.7$ is shown in figure 1.6. The solid line represents the objective function with $N = 64$ and the dotted line is the objective function computed with $N = 32$. The vertical lines in the figure are artifacts of the plotting package; no lines should be connecting the stairs of the function. Here we see that the discontinuity in the velocity distribution causes discontinuities in the objective function. Further, the number of discontinuities is dependent on the discretization of the flow solution. In figure 1.7, the terms in (1.24) are plotted against i for $\xi = 1.2$ and $N = 64$. The summation is dominated by the differences in the target and calculated velocity in the segment between shocks. For small perturbations of the design variable, provided the shock remains between the same grid points, the value of the objective function changes very little.

These perturbations occur along a single stair in figure 1.6. For perturbations of the design variable which cause the shock to move across a grid point, the objective function changes dramatically as another dominant term is added or removed in the summation. These perturbations cause the jumps in figure 1.6. The jumps are less frequent in the case with $N = 32$ than with $N = 64$ because there are fewer grid points for the shock wave to pass through.

In the case of the objective function computed using the exact analysis, the velocity distribution contains a discontinuity and so the objective function is discontinuous as well. The Godunov scheme smears the shock wave over two grid cells and thus the objective function is not discontinuous. Yet as can be seen in figure 1.8, the function is non-smooth and contains regions of local minima. As the resolution of the shock wave diminishes, the objective function becomes smoother. In figure 1.9, the objective function has been computed using the artificial viscosity solver with $\alpha = 1$. The shock wave is smeared enough so that the objective function appears well-behaved.

In CFD design it is not desirable to obtain a well-behaved objective function at the expense of the quality of the flow solution. One way to eliminate the non-smoothness of the objective function is to perform a more exact integration of (1.22). This involves first dividing the integral at the location of the discontinuities and then applying the trapezoidal approximation to each segment. The objective function contains two steep gradient stemming from the target and the computed shock, and thus the integral is divided into three segments

$$I(\xi) = \frac{1}{2}\int_0^{\hat{x}_s^-} (\hat{u} - u)^2 dx + \frac{1}{2}\int_{\hat{x}_s^+}^{x_s^-} (\hat{u} - u)^2 dx + \frac{1}{2}\int_{x_s^+}^1 (\hat{u} - u)^2 dx, \qquad (1.25)$$

where $\hat{x}_s$ and x_s are the target and calculated shock positions respectively. Equation (1.25) is valid if $\hat{x}_s < x_s$, otherwise $\hat{x}_s$ and x_s must be interchanged. Further

insight reveals that since only the computed shock wave position interacts with the discretization, ($\hat{x}_s$ remains constant throughout the optimization) we only need to divide (1.22) into two parts to get a smoother objective function

$$I(\xi) = \frac{1}{2}\int_0^{x_s^-} (\hat{u} - u)^2 dx + \frac{1}{2}\int_{x_s^+}^1 (\hat{u} - u)^2 dx. \tag{1.26}$$

This is called the shock-fitted objective function. Performing the integration of (1.26) numerically requires that grid points be placed on either side of the shock. Using the exact analysis, the position of the shock as well as the left and right values of the velocity at the shock is known. The smooth shock-fitted objective function computed using the exact analysis is drawn in figure 1.10.

Using numerical solutions, the shock position and left and right values of the shock are not clearly defined quantities. For the purposes of computing the shock-fitted objective function with numerical solutions we fabricate definitions for the quantities. The shock position is defined as the location where the velocity distribution crosses the sonic line. This position is found by interpolating between the grid points where the crossover from supersonic to subsonic flow takes place. We have

$$x_s = x_{js} + \frac{u_* - u_{js}}{u_{js+1} - u_{js}}(x_{js+1} - x_{js}), \tag{1.27}$$

where the subscripts js and $js+1$ are the grid index before and after the shock. The sonic velocity is defined by

$$u_* = \sqrt{2h_o \frac{\gamma - 1}{\gamma + 1}}, \tag{1.28}$$

The left and right values of the velocity are computed by left and right extrapolations to the shock position. Care must be taken when extrapolating so that information is not taken from a point "trapped" in a shock. A clear picture of the interpolation and extrapolation is shown in figure 1.11.

These definitions do not remove all the noise generated by the interaction of the shock wave and the discretization of the flow field yet there is an improvement in the smoothness of the objective function and the regions of local minima are gone. The shock-fitted objective function plotted with the Godunov solutions is shown in figure 1.12. A comparison to figure 1.8 shows the improvement.

The extrapolation for left and right velocities of smeared solutions such as those of the artificial viscosity with $\alpha = 1$ has little meaning due to the severity of the dissipation surrounding the shock. Further, the interaction of the shock wave with the discretization is not severe in such cases. For this reason, the shock-fitted objective function is not computed using highly dissipative schemes.

The shock-fitted technique is not the only way to obtain a smooth objective function. Another elegant method strains the computed flow solution using a coordinate transformation so that the shock position is aligned with the shock position of the target. This method of coordinate straining was developed by Nixon[65] and used in transonic airfoil design by Joh.[61]

The implementation of coordinate straining involves defining a straining function, $s(x)$, which equals 0 at the inlet and exit, and has a value of 1 at the position of the target shock. The function is not unique. From reference 67, we choose

$$s(x) = \left(\frac{x}{\hat{x}_s}\right)\left(\frac{1-x}{1-\hat{x}_s}\right). \tag{1.29}$$

The calculated velocity distribution is strained proportionately to the distance between shocks according to

$$\tilde{u} = u(x - s\Delta x_s), \tag{1.30}$$

where $\tilde{u}$ is the strained velocity and $\Delta x_s \equiv \hat{x}_s - x_s$ is the distance between computed and target shock waves. For a numerical solution we apply the transformation to a discrete set of points representing the velocity distribution on a mesh, we determine

a grid index, M, such that $x_M < x_i - s\Delta x_s < x_{M+1}$. Then using linear interpolation, the strained velocity can be computed as

$$\tilde{u}_i = u_M + \frac{x_i - s\Delta x_s - x_M}{\Delta x}(u_{M+1} - u_M). \tag{1.31}$$

The coordinate straining transformation is shown in figure 1.13.

The evaluation of the objective function now uses $\tilde{u}$ in place of u, removing the dominating terms which exist in the region between the calculated and target shocks. In the process of removing the non-smoothness, the objective function has become very flat near the minimum. Using this technique we rely on the small differences in velocities to drive the area to the target. Often this is enough to improve the design, but not enough to achieve the best possible one.

To shape the objective function to better define the minimum, a shock penalty proportional to the square of the difference between shock waves is added to the strained objective function, yielding

$$I(\xi) = \frac{1}{2}\sum_{i=2}^{N-1}(\hat{u} - \tilde{u})_i^2 \Delta x + \frac{1}{2}\sigma \Delta x_s^2, \tag{1.32}$$

where σ is a positive constant. Values of σ can be arbitrarily chosen or defined so that (1.32) equals (1.24) during the first iteration of optimization. Work presented here used $\sigma = 5$. Figure 1.14, 1.15, and 1.16., show the coordinate strained/shock penalty objective using the three flow solution methods.

CHAPTER
TWO

THE OPTIMIZATION PROCEDURE

In this chapter a first-order optimization method is outlined for minimizing the unconstrained problems mentioned in chapter 1. The goal of the optimization study is to compare several different methods for computing design derivatives. The optimization technique acts as a vehicle for making the comparison. For this reason the technique chosen is simple, yet robust, so that failure of an optimization can be easily traceable. In this way we can determine if the design derivative methods are the cause of the failure or not.

Consider the general unconstrained minimization problem in n-dimensional space

$$\min_{\xi \in \Re^n} I(\xi), \tag{2.1}$$

where I is a differentiable function. One way to solve (2.1) iteratively is to take steps in the design space in the direction of a descent. For example, a step of size α ($\alpha > 0$) from the point ξ_o in the direction of p_o leads to $I(\xi_o + \alpha p_o/\|p_o\|)$ which can be approximated by the two term Taylor Series

$$I(\xi_o + \alpha \frac{p_o}{\|p_o\|}) \simeq I(\xi_o) + \alpha \nabla I^T(\xi_o) \frac{p_o}{\|p_o\|}. \tag{2.2}$$

It becomes obvious from (2.2) that choosing p_o so that $\alpha \nabla I^T(\xi_o) p_o/\|p_o\|$ is large and negative will lead to a large reduction in the function. Thus the aim is to minimize $\alpha \nabla I^T(\xi_o) p_o/\|p_o\|$. Since α is a positive scalar, the problem becomes

$$\min_{p_o \in \Re^n} \nabla I^T(\xi_o) \frac{p_o}{\|p_o\|}. \tag{2.3}$$

The solution is dependent on the choice of norms. For the l_2 norm we solve

$$\min_{p_o \in \Re^n} \nabla I^T(\xi_o) \frac{p_o}{(p_o^T p_o)^{1/2}}, \tag{2.4}$$

and the solution is the negative gradient.[47] Thus the negative gradient vector represents the direction of steepest descent for the objective function.

The algorithm for finding the optimum through the direction of steepest descent is outlined as follows:

- provide an initial point and evaluate the objective function at this point,
- test for convergence,
- compute the gradient of the function,
- choose a step size so that $I(\xi_k + \alpha p_k) < I(\xi_k)$,*
- update the solution with $\xi_{k+1} = \xi_k + \alpha p_k$.

The method seems effective, yet suffers from poor convergence rates. Its inefficiencies stem from the fact that no second order information, nor any information from previous iterations, is used to determine the descent direction. An example of optimization using the steepest descent method is shown graphically in figure 2.1. The example solves

$$\min_{p \in \Re^2} I(\xi_1, \xi_2) = 10\xi_1^2 + \xi_2^2 \tag{2.5}$$

The function was minimized to 10^{-12} in 58 iterations starting from the point (10,2). Due to the nature of the steepest descent algorithm, the convergence slows considerably near the minimum.

For the purpose of comparing design sensitivity algorithms, the method of steepest descent is sufficient, however a slight modification to the algorithm can lead to big payoffs in the convergence rate. To improve the convergence while avoiding

*To guarantee convergence with this algorithm, a stronger requirement such as the Goldstein-Armijo principle[47] is needed here.

to store or approximate the Hessian is to force the descent direction to be conjugate to the directions of previous iterations. A conjugate direction is one in which the change in the gradient is perpendicular to the previous descent direction.[47] This is the method of conjugate gradients and is the method used in this work. The first descent direction chosen is the steepest descent direction. All others follow the formula[68]

$$p_k = -\nabla I_k + \beta_{k-1} p_{k-1}, \tag{2.6}$$

where

$$\beta_{k-1} = \frac{\|\nabla I_k\|_2^2}{\|\nabla I_{k-1}\|_2^2}. \tag{2.7}$$

In theory, the method of conjugate gradients should find the minimum in n steps or less for an n-variable quadratic function. However, roundoff errors associated with the computation of the derivatives can quickly cause the descent directions to lose conjugacy. Thus more than n iterations may be required to find the minimum. In our algorithm, after n iterations, the method is restarted with a search in the steepest descent direction to regain conjugacy. The algorithm is outlined as follows

- provide an initial point and evaluate the objective function at this point,
- test for convergence,
- compute the descent direction according to equation (2.6) with restarts after n iterations,
- choose a step size so that $I(\xi_k + \alpha p_k) < I(\xi_k)$,
- update the solution with $\xi_{k+1} = \xi_k + \alpha p_k$.

The minimization of (2.5) is repeated using the conjugate gradient method. The function is minimized to 10^{-12} in 3 iterations. Figure 2.2 graphically shows the minimization process.

In the optimization procedures outlined, we must choose a suitable step size so that $I(\xi_k + \alpha p_k) < I(\xi_k)$. There are many procedures available for finding

the optimal step size. Results presented in this dissertation use Brent's method[69] coded in FORTRAN by Press, Teukolsky, Vetteling, and Flannery.[70] Essentially, the method fits a quadratic through three data points in an interval containing the minimum. The formula for the location of minimum of the quadratic through the points $(a, f(a))$, $(b, f(b))$, $(c, f(c))$ is given by

$$x = b - \frac{1}{2}\frac{(b-a)^2[f(b)-f(c)] - (b-c)^2[f(b)-f(a)]}{(b-a)[f(b)-f(c)] - (b-c)[f(b)-f(a)]}. \tag{2.8}$$

The algorithm guards against x being a minimum and not a maximum. Several curve fits are performed to find the exact minimum in the one-dimensional search.

CHAPTER
THREE

CONTINUOUS APPROACH TO DERIVATIVE CALCULATIONS

The optimization routine outlined in the previous chapter requires the derivatives of the objective function with respect to the design variables. The most straight forward way to estimate the derivatives is to use finite differences. A first-order forward difference is

$$\frac{\partial I}{\partial \xi_j} \cong \frac{1}{\Delta \xi_j} \left[I(\xi_1, \xi_2, \ldots, \xi_j + \Delta \xi_j, \ldots, \xi_n) - I(\xi_1, \xi_2, \ldots, \xi_n) \right], \tag{3.1}$$

and a second-order central difference is

$$\frac{\partial I}{\partial \xi_j} \cong \frac{1}{2\Delta \xi_j} \left[I(\xi_1, \xi_2, \ldots, \xi_j + \Delta \xi_j, \ldots, \xi_n) - I(\xi_1, \xi_2, \ldots, \xi_j - \Delta \xi_j, \ldots, \xi_n) \right]. \tag{3.2}$$

These methods are inefficient. In each iteration of the optimization routine, $n + 1$ function evaluations are required to compute the first-order derivatives, or $2n$ function evaluations for second-order derivatives. Each function evaluation requires a flow solution which for many fluid dynamics problems can be prohibitively expensive. Many cheaper methods to compute the derivatives are available and are the topic of this chapter and the next.

We investigate two approaches for computing the derivatives. The first is the continuous approach which is formulated from equations in their continuous form and is discussed in this chapter. In the next chapter we investigate the discrete approach which formulates the derivatives from discretized equations. In each approach the direct and adjoint methods for computing derivatives are derived.

3.1 Direct Method

In the continuous approach the shock-fitted objective function defined by (1.26) is utilized. Analytic differentiation of (1.26) with respect to the j^{th} design variable requires Liebnitz' rule as the shock position is a function of the design variables

$$\frac{dI}{d\xi} = -\int_0^{x_s^-} (\hat{u}-u)u'dx - \int_{x_s^+}^1 (\hat{u}-u)u'dx + x_s'\left[\frac{(\hat{u}-u)^2}{2}\right]_{x=x_s^-} - x_s'\left[\frac{(\hat{u}-u)^2}{2}\right]_{x=x_s^+}. \tag{3.3}$$

Some explanation of the notation is in order. For simplicity, the subscript j has been dropped. Realize that (3.3) can be written for each design variable $\xi_1, \xi_2, \ldots, \xi_n$. When we utilize the resulting formula, we will interpret $\partial/\partial\xi$ as $\partial/\partial\xi_j$ with all other variables held fixed. The primes indicate differentiation with respect to ξ_j so that $x_s' = dx_s/d\xi_j$ and $u' = \partial u/\partial\xi_j$ with x held constant. The subscripts on the square bracket terms denote where the terms are evaluated.

Equation (3.3) can be used to evaluate the derivatives provided expressions for u' and x_s' can be found. In the continuous direct method, we will obtain an ordinary differential equation which can be solved for u' and an algebraic equation for x_s'. The ordinary differential equation comes from differentiating the steady-state form of the governing equation (1.3) with respect to ξ. In (1.3), $f = f(u)$ and $g = g(u,\xi)$ are given by (1.4) and (1.5) respectively. We differentiate (1.3) utilizing the chain rule and by interchanging the order of the x and ξ derivatives we obtain

$$a\frac{\partial u'}{\partial x} + \left(\frac{da}{du}\frac{\partial u}{\partial x} + b\right)u' + \frac{\partial g}{\partial \xi} = 0, \tag{3.4}$$

where $a \equiv df/du$ and $b \equiv \partial g/\partial u$. Note that (3.4) is only valid where the solution, u, is continuous. This linear ordinary differential equation for $u'(x)$ we call the direct equation. Analytic expressions for the coefficients can be derived from (1.4) and (1.5)

$$a = 1 - \frac{\gamma-1}{\gamma+1}\frac{2h_o}{u^2}, \tag{3.5}$$

$$\frac{da}{du} = \frac{\gamma - 1}{\gamma + 1} \frac{4h_o}{u^3}, \tag{3.6}$$

$$b = \frac{A_x}{A} \frac{\gamma - 1}{\gamma + 1} \left(1 + \frac{2h_o}{u^2} \right), \tag{3.7}$$

and

$$\frac{\partial g}{\partial \xi} = \frac{\gamma - 1}{\gamma + 1} \left(u - \frac{2h_o}{u} \right) \frac{\partial}{\partial x} (\ln A'). \tag{3.8}$$

We can write the governing equation (1.3) as $a\partial u/\partial x + g = 0$ so that

$$\frac{\partial u}{\partial x} = -\frac{g}{a}. \tag{3.9}$$

To complete the boundary value problem we differentiate the boundary conditions with respect to ξ. At the inlet and outlet we have

$$u'(0, \xi) = 0, \tag{3.10}$$

and

$$u'(1, \xi) = 0, \tag{3.11}$$

The direct equation may be solved analytically using integrating factors. Differentiating Du' with respect to x, where D is the integrating factor yields

$$\frac{d(Du')}{dx} = D\frac{\partial u'}{\partial x} + \frac{dD}{dx}u', \tag{3.12}$$

or

$$\frac{\partial u'}{\partial x} + \frac{1}{D}\frac{dD}{dx}u' - \frac{1}{D}\frac{d(Du')}{dx} = 0 \tag{3.13}$$

Comparing (3.13) to the direct equation we see that

$$\frac{1}{D}\frac{dD}{dx} = \frac{1}{a}\left(\frac{da}{du}\frac{\partial u}{\partial x} + b \right), \tag{3.14}$$

and

$$\frac{1}{D}\frac{d(Du')}{dx} = -\frac{1}{a}\frac{\partial g}{\partial \xi}. \tag{3.15}$$

From (3.15) we find

$$d(Du') = -\frac{D}{a}\frac{\partial g}{\partial \xi}dx. \tag{3.16}$$

We integrate from both ends using boundary conditions (3.10) and (3.11) to get u' as a function of the integrating factor,

$$u'(x) = -\frac{1}{D(x)}\int_0^x \frac{D}{a}\frac{\partial g}{\partial \xi}d\chi, \qquad 0 \le x < x_s, \tag{3.17}$$

and

$$u'(x) = \frac{1}{D(x)}\int_x^1 \frac{D}{a}\frac{\partial g}{\partial \xi}d\chi, \qquad x_s \le x < 1. \tag{3.18}$$

From (3.14) we solve for D,

$$\frac{dD}{D} = \frac{1}{a}\left(\frac{da}{du}\frac{\partial u}{\partial x} + b\right)dx. \tag{3.19}$$

Again, we integrate from both ends

$$D(x) = D(0)\exp\left\{\int_0^x \frac{1}{a}\left(\frac{da}{du}\frac{\partial u}{\partial x} + b\right)d\chi\right\}, \qquad 0 \le x < x_s. \tag{3.20}$$

and

$$D(x) = D(1)\exp\left\{-\int_x^1 \frac{1}{a}\left(\frac{da}{du}\frac{\partial u}{\partial x} + b\right)d\chi\right\}, \qquad x_s \le x < 1. \tag{3.21}$$

Equations (3.17) and (3.18) with (3.20) and (3.21) form the solution to the direct equation. It can easily be seen by substituting (3.20) into (3.17) that the value of the integrating factor at $x = 0$ drops out of the equation. A similar argument holds for $D(1)$ with equations (3.18) and (3.21).

To find an expression for x_s', we differentiate the shock jump condition (1.14). This yields

$$u^-\frac{du^+}{d\xi} + u^+\frac{du^-}{d\xi} = 0. \tag{3.22}$$

Note that $u^{\pm} = u(x_s^{\pm}, \xi)$. Then by the chain rule

$$\frac{du^{\pm}}{d\xi} = \left(\frac{\partial u}{\partial \xi} + x_s' \frac{\partial u}{\partial x} \right)_{x=x_s^{\pm}}. \tag{3.23}$$

We will adopt the notation $(u^{\pm})'$ to be $\partial u/\partial \xi$ taken at constant x and evaluated at $x = x_s^{\pm}$.

Note that in (3.23) $\partial u/\partial \xi$ evaluated at $x = x_s^{\pm}$ is discontinuous, with either the left or right derivatives being infinite for a sharp shock. Equation (3.23) is valid only for the one-sided finite derivative. That is

$$\left(\frac{\partial u}{\partial \xi} \right)_{x=x_s^+} = \lim_{\varepsilon \to 0} \left(\frac{\partial u}{\partial \xi} \right)_{x=x_s+\varepsilon}, \tag{3.24}$$

with a similar definition for $\partial u/\partial \xi$ at $x = x_s^-$. We have ensured that our numerical approximation of these derivatives have taken the above definitions into account.

We can combine (3.22) and (3.23) to develop and expression for x_s'

$$x_s' = -\frac{u^-(u^+)' + u^+(u^-)'}{u^- \frac{\partial u}{\partial x}\Big|_{x=x_s^+} + u^+ \frac{\partial u}{\partial x}\Big|_{x=x_s^-}}. \tag{3.25}$$

In the continuous direct method we calculate the design derivatives using (3.3). The u' terms are calculated as a solution to the direct equation (3.4) subject to the boundary conditions (3.10) and (3.11). The x_s' term in (3.3) is evaluated from (3.25) with the $(u^{\pm})'$ terms evaluated from u' at $x = x_s^{\pm}$.

In the direct methods we eliminate the burden of computing at least $n+1$ flow solutions, but we must solve equation (3.4) n times. For stability purposes, the flow equation (1.3) is solved numerically by adding an unsteady term and marching to a steady state. Equation (3.4) has the advantage of being a linear system and can be solved numerically or analytically as an ordinary differential equation. Because of its linearity the solution to the direct equation is computationally cheaper than the flow solution.

3.1 Adjoint Method

The adjoint method avoids computing $u'(x)$ and x_s' directly. This method defines an augmented functional

$$\begin{aligned} I^*(\xi) =& \frac{1}{2}\int_0^{x_s^-} (\hat{u}-u)^2 dx + \frac{1}{2}\int_{x_s^+}^1 (\hat{u}-u)^2 dx \\ &+ \int_0^{x_s^-} \lambda_1(\frac{\partial f}{\partial x}+g)dx + \int_{x_s^+}^1 \lambda_2(\frac{\partial f}{\partial x}+g)dx \\ &+ \lambda_3\left[u^-u^+ - 2\frac{(\gamma-1)}{(\gamma+1)}h_o\right] \\ &+ \lambda_4\left[u_0-u(0)\right] + \lambda_5\left[u_1-u(1)\right], \end{aligned} \tag{3.26}$$

where $\lambda_1, \ldots, \lambda_5$ are Lagrange multipliers. As is customary, the augmented functional is defined by adding terms which are identically zero. Here λ_1 and λ_2 multiply the governing equation over the region where it is valid. For the region at the shock, we include the shock jump relation. The final terms in the augmented functional represent boundary conditions for the governing equation. To find the derivative of the augmented functional, again we apply Liebnitz' rule to obtain

$$\begin{aligned} \frac{dI^*}{d\xi} =& -\int_0^{x_s^-} (\hat{u}-u)u'dx + x_s'\left[\frac{(\hat{u}-u)^2}{2}\right]_{x=x_s^-} \\ &- \int_{x_s^+}^1 (\hat{u}-u)u'dx - x_s'\left[\frac{(\hat{u}-u)^2}{2}\right]_{x=x_s^+} \\ &+ \int_0^{x_s^-} \lambda_1(\frac{\partial f'}{\partial x}+g')dx + x_s'\left[\lambda_1(\frac{\partial f}{\partial x}+g)\right]_{x=x_s^-} \\ &+ \int_{x_s^+}^1 \lambda_2(\frac{\partial f'}{\partial x}+g')dx - x_s'\left[\lambda_2(\frac{\partial f}{\partial x}+g)\right]_{x=x_s^+} \\ &+ \lambda_3\left(u^-(u^+)' + (u^-)'u^+\right) - \lambda_4 u'(0) - \lambda_5 u'(1). \end{aligned} \tag{3.27}$$

With the definition of the augmented functional, the design derivative $dI/d\xi$ will be equal to $dI^*/d\xi$. In the adjoint method we choose the multipliers such that u' and

x_s' will not appear in the expression for $dI^*/d\xi$. First we perform some operations to simplify (3.27).

The square bracket terms in (3.27) containing λ_1 and λ_2 are identically zero since the governing equation is zero over the region where they are valid. Over continuous regions we can integrate by parts the integrals involving λ_1 as

$$\int \lambda_1 \left(\frac{\partial f'}{\partial x} + g' \right) dx = \int \left[\frac{\partial(\lambda_1 f')}{\partial x} + \lambda_1 g' \right] dx - \int \frac{\partial \lambda_1}{\partial x} f' dx. \tag{3.28}$$

Likewise for λ_2 we have

$$\int \lambda_2 \left(\frac{\partial f'}{\partial x} + g' \right) dx = \int \left[\frac{\partial(\lambda_2 f')}{\partial x} + \lambda_2 g' \right] dx - \int \frac{\partial \lambda_2}{\partial x} f' dx. \tag{3.29}$$

Using the chain rule for $f = f(u)$ and $g = g(u, \xi)$ we can determine $f' = au'$ and $g' = \partial g/\partial \xi + bu'$. Substituting these relations into (3.27) and evaluating exact differentials we get

$$\begin{aligned}
\frac{dI^*}{d\xi} = & - \int_0^{x_s^-} \left[a\frac{\partial \lambda_1}{\partial x} - b\lambda_1 + (\hat{u} - u) \right] u' dx + \int_0^{x_s^-} \lambda_1 \frac{\partial g}{\partial \xi} dx \\
& - \int_{x_s^+}^1 \left[a\frac{\partial \lambda_2}{\partial x} - b\lambda_2 + (\hat{u} - u) \right] u' dx + \int_{x_s^+}^1 \lambda_2 \frac{\partial g}{\partial \xi} dx \\
& + \lambda_1 f'|_{x=x_s^-} - \lambda_1 f'|_{x=0} + \lambda_2 f'|_{x=1} - \lambda_2 f'|_{x=x_s^+} \\
& x_s' \left[\left(\frac{(\hat{u} - u)^2}{2} \right)_{x=x_s^-} - \left(\frac{(\hat{u} - u)^2}{2} \right)_{x=x_s^+} \right] \\
& + \lambda_3 \left(u^-(u^+)' + u^+(u^-)' \right) - \lambda_4 u'(0) - \lambda_5 u'(1).
\end{aligned} \tag{3.30}$$

Again we apply (3.22) and (3.23) to the derivative calculation. This allows us to group terms multiplying $(u^-)'$, $(u^+)'$, and x_s'

$$\begin{aligned}
\frac{dI}{d\xi} = & - \int_0^{x_s^-} \left[a\frac{\partial \lambda_1}{\partial x} - b\lambda_1 + (\hat{u} - u) \right] u' dx + \int_0^{x_s^-} \lambda_1 \frac{\partial g}{\partial \xi} dx \\
& - \int_{x_s^+}^1 \left[a\frac{\partial \lambda_2}{\partial x} - b\lambda_2 + (\hat{u} - u) \right] u' dx + \int_{x_s^+}^1 \lambda_2 \frac{\partial g}{\partial \xi} dx \\
& + \left[\lambda_3 u^+ + \lambda_1^- a^- \right] (u^-)' + \left[\lambda_3 u^- - \lambda_2^+ a^+ \right] (u^+)'
\end{aligned}$$

$$+ x_s' \left[\left(\frac{(\hat{u}-u)^2}{2} \right)_{x=x_s^-} - \left(\frac{(\hat{u}-u)^2}{2} \right)_{x=x_s^+} \right.$$
$$\left. + \lambda_3 \left(u^- \left. \frac{\partial u}{\partial x} \right|_{x=x_s^+} + u^+ \left. \frac{\partial u}{\partial x} \right|_{x=x_s^-} \right) \right]$$
$$+ \lambda_2 a u' |_{x=1} - \lambda_1 a u' |_{x=0} - \lambda_4 u'(0) - \lambda_5 u'(1). \tag{3.31}$$

The adjoint problem is specified by picking values for the Lagrange multipliers so that certains terms in (3.31) are zero. In particular we choose the multipliers such that u', $(u^-)'$, $(u^+)'$, and x_s' do not appear in the formulation. Specifically, we enforce

$$a \frac{\partial \lambda_1}{\partial x} - b\lambda_1 + (\hat{u} - u) = 0 \quad on \quad 0 \le x \le x_s^- \tag{3.32}$$

$$a \frac{\partial \lambda_2}{\partial x} - b\lambda_2 + (\hat{u} - u) = 0 \quad on \quad x_s^+ \le x \le 1 \tag{3.33}$$

$$\lambda_3 = \frac{\left(\frac{(\hat{u}-u)^2}{2} \right)_{x=x_s^+} - \left(\frac{(\hat{u}-u)^2}{2} \right)_{x=x_s^-}}{\left(u^- \left. \frac{\partial u}{\partial x} \right|_{x=x_s^+} + u^+ \left. \frac{\partial u}{\partial x} \right|_{x=x_s^-} \right)}, \tag{3.34}$$

$$\lambda_3 u^+ + \lambda_1^- a^- = 0, \tag{3.35}$$

$$\lambda_3 u^- - \lambda_2^+ a^+ = 0. \tag{3.36}$$

Equations (3.32) through (3.36) make up the adjoint problem. Like the direct problem, equations (3.32) and (3.33) along with boundary conditions (3.35) and (3.36) can be solved analytically. We find that utilizing an appropriate integrating factor

$$\lambda_1(x) = \frac{1}{R_1} \left\{ \lambda_1^- + \int_x^{x_s^-} \frac{(\hat{u} - u)}{a} R_1 \, d\chi \right\}, \tag{3.37}$$

on $0 \le x \le x_s^-$ where

$$R_1(x) = \exp \left\{ \int_x^{x_s^-} \frac{b}{a} d\chi \right\}, \tag{3.38}$$

and

$$\lambda_2(x) = \frac{1}{R_2} \left\{ \lambda_2^+ - \int_{x_s^+}^{x} \frac{(\hat{u} - u)}{a} R_2 \, d\chi \right\}, \tag{3.39}$$

on $x_s^+ \leq x \leq 1$ where

$$R_2(x) = \exp\left\{-\int_{x_s^+}^{x} \frac{b}{a} d\chi\right\}. \tag{3.40}$$

In this work the integrals solving the direct and adjoint equations are evaluated using a trapezoidal approximation.

As in the continuous direct method $\partial u/\partial x$ evaluated to the right and left of the shock is computed using $\partial u/\partial x = -g/a$ at $x = x_s^{\pm}$. Under the conditions specified by the adjoint problem, the derivative becomes

$$\frac{\partial I^*}{\partial \xi} = \int_0^{x_s^-} \lambda_1 \frac{\partial g}{\partial \xi} dx + \int_{x_s^+}^{1} \lambda_2 \frac{\partial g}{\partial \xi} dx. \tag{3.41}$$

We see that in the continuous adjoint method we must solve one BVP for $u(x, \xi)$ and one BVP for the Lagrange multipliers. The derivatives may then be calculated from (3.41) by quadrature. In the direct method, recall that we solved one BVP for u and n BVP for u'. In general, adjoint methods will be computationally cheaper than direct methods.

CHAPTER
FOUR

DISCRETE APPROACH TO DERIVATIVE CALCULATIONS

In the previous chapter, the direct and adjoint methods were applied to the quasi-one-dimensional duct design problem using the continuous approach and the shock-fitted objective function. The alternative is to apply the method in a discrete sense. That is the direct and adjoint methods are applied to the discretized equations. Here we make the formulation using the coordinate-strained and shock-penalty objective function.

4.1 Direct Method

In the discrete methods we no longer consider the velocity to a be a function of x, rather it is a vector of dimension N whose elements are the velocity values at the cell centers of the flow domain. To evaluate the objective function the velocity is strained according to (1.30) to obtain the vector $\tilde{u}$. The coordinate-strained objective function with the shock penalty then has the functional dependence

$$I = I[\tilde{u}(\xi), x_s(\xi)], \tag{4.1}$$

where

$$\tilde{u} = [\,\tilde{u}_1 \quad \tilde{u}_2 \quad \dots \quad \tilde{u}_N\,]^T. \tag{4.2}$$

Applying the chain rule to (4.1) to compute the derivative, we have

$$\frac{\partial I}{\partial \xi_j} = \frac{\partial I}{\partial \tilde{u}} \frac{\partial \tilde{u}}{\partial \xi_j} + \frac{\partial I}{\partial x_s} \frac{\partial x_s}{\partial \xi_j}, \tag{4.3}$$

where

$$\frac{\partial I}{\partial \tilde{u}} = \left[\frac{\partial I}{\partial \tilde{u}_1} \quad \frac{\partial I}{\partial \tilde{u}_2} \quad \cdots \quad \frac{\partial I}{\partial \tilde{u}_N} \right], \tag{4.4}$$

and

$$\frac{\partial \tilde{u}}{\partial \xi_j} = \left[\frac{\partial \tilde{u}_1}{\partial \xi_j} \quad \frac{\partial \tilde{u}_2}{\partial \xi_j} \quad \cdots \quad \frac{\partial \tilde{u}_N}{\partial \xi_j} \right]^T. \tag{4.5}$$

Each element in (4.4) has the form

$$\frac{\partial I}{\partial \tilde{u}_i} = -(\hat{u} - \tilde{u})_i \Delta x, \tag{4.6}$$

and $\partial I / \partial x_s$ is simply

$$\frac{\partial I}{\partial x_s} = -\sigma(\hat{x}_s - x_s). \tag{4.7}$$

Equation (4.5) can be expanded further. The straining function (1.30) is a function of u and x_s, so that the dependence is

$$\tilde{u} = \tilde{u}[u(\xi), x_s(\xi)]. \tag{4.8}$$

Differentiation with respect to the jth design variable yields

$$\frac{\partial \tilde{u}}{\partial \xi_j} = \frac{\partial \tilde{u}}{\partial u} \frac{\partial u}{\partial \xi_j} + \frac{\partial \tilde{u}}{\partial x_s} \frac{\partial x_s}{\partial \xi_j}, \tag{4.9}$$

where

$$\frac{\partial \tilde{u}}{\partial u} = \begin{bmatrix} \frac{\partial \tilde{u}_1}{\partial u_1} & \frac{\partial \tilde{u}_1}{\partial u_2} & \cdots & \frac{\partial \tilde{u}_1}{\partial u_N} \\ \frac{\partial \tilde{u}_2}{\partial u_1} & \frac{\partial \tilde{u}_2}{\partial u_2} & \cdots & \frac{\partial \tilde{u}_2}{\partial u_N} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial \tilde{u}_N}{\partial u_1} & \frac{\partial \tilde{u}_N}{\partial u_2} & \cdots & \frac{\partial \tilde{u}_N}{\partial u_N} \end{bmatrix}, \tag{4.10}$$

$$\frac{\partial u}{\partial \xi_j} = \left[\frac{\partial u_1}{\partial \xi_j} \quad \frac{\partial u_2}{\partial \xi_j} \quad \cdots \quad \frac{\partial u_N}{\partial \xi_j} \right]^T, \tag{4.11}$$

and

$$\frac{\partial \tilde{u}}{\partial x_s} = \left[\frac{\partial \tilde{u}_1}{\partial x_s} \quad \frac{\partial \tilde{u}_2}{\partial x_s} \quad \cdots \quad \frac{\partial \tilde{u}_N}{\partial x_s} \right]^T. \tag{4.12}$$

Equation (4.3) can now be written as

$$\frac{\partial I}{\partial \xi_j} = \frac{\partial I}{\partial \tilde{u}} \left(\frac{\partial \tilde{u}}{\partial u} + \frac{\partial \tilde{u}}{\partial x_s} \frac{\partial x_s}{\partial u} \right) \frac{\partial u}{\partial \xi_j} + \frac{\partial I}{\partial x_s} \frac{\partial x_s}{\partial \xi_j}. \tag{4.13}$$

The Jacobian $\partial\hat{u}/\partial u$ is sparse with the elements defined according to

$$\frac{\partial \tilde{u}_i}{\partial u_j} = \begin{cases} 0 & i,j = 1,\ldots,N; \quad j \neq M, M+1 \\ 1 - \frac{x_i - s_i \Delta x_s - x_M}{\Delta x} & i = 1,\ldots,N; \quad j = M \\ \frac{x_i - s_i \Delta x_s - x_M}{\Delta x} & i = 1,\ldots,N; \quad j = M \end{cases}, \tag{4.14}$$

where M is the grid index such that $x_M < x_j - s\Delta x_s < x_{M+1}$. Also according to the straining function we find

$$\frac{\partial \tilde{u}_i}{\partial x_s} = \frac{s_i}{\Delta x}(u_{M+1} - u_M), \tag{4.15}$$

where s_i is the discretized version of (1.29), *i.e.*

$$s_i = \left(\frac{x_i}{\hat{x}_s}\right)\left(\frac{1 - x_i}{1 - \hat{x}_s}\right). \tag{4.16}$$

We can also expand the $\partial x_s/\partial \xi_j$ term appearing in (4.13). The shock position is defined by (1.27) and has the functional dependence

$$x_s = x_s[u(\xi)]. \tag{4.17}$$

The shock position is dependent on ξ_j through the velocities of the surrounding cells. However, in general, we may write

$$\frac{\partial x_s}{\partial \xi_j} = \frac{\partial x_s}{\partial u}\frac{\partial u}{\partial \xi_j}, \tag{4.18}$$

where

$$\frac{\partial x_s}{\partial u} = \left[\, \frac{\partial x_s}{\partial u_1} \quad \frac{\partial x_s}{\partial u_2} \quad \cdots \quad \frac{\partial x_s}{\partial u_N} \,\right]. \tag{4.19}$$

Realize that only two components $\partial x_s/\partial u$ of will be non zero

$$\frac{\partial x_s}{\partial u_i} = \begin{cases} 0 & i = 1,\ldots,N; \quad i \neq js, js+1 \\ -\frac{(u_* - u_{js})}{(u_{js+1} - u_{js})^2}\Delta x & i = js \\ -\frac{(u_* - u_{js})}{(u_{js+1} - u_{js})^2}\Delta x - \frac{\Delta x}{(u_{js+1} - u_{js})} & i = js+1 \end{cases}. \tag{4.20}$$

Substituting (4.9) and (4.18) into (4.3) gives us an expression for the design derivatives

$$\frac{\partial I}{\partial \xi_j} = v^T \frac{\partial u}{\partial \xi_j}, \tag{4.21}$$

where

$$v^T \equiv \frac{\partial I}{\partial \tilde{u}} \left(\frac{\partial \tilde{u}}{\partial u} + \frac{\partial \tilde{u}}{\partial x_s} \frac{\partial x_s}{\partial u} \right) + \frac{\partial I}{\partial x_s} \frac{\partial x_s}{\partial u}. \tag{4.22}$$

Every term in (4.22) is known analytically, namely (4.6), (4.7), (4.10), (4.15), and (4.19). What remains to be found is an expression of $\partial u / \partial \xi_j$ in (4.21). The discrete direct method applies the chain rule to the governing equations discretized by some numerical scheme. We have N discretized equations that are, in general, functions of the velocity and the design variables. These equations are given the symbol w

$$\begin{aligned} w_1 &= w_1(u_1, \ldots, u_N, \xi_1, \ldots \xi_n) = 0 \\ w_2 &= w_2(u_1, \ldots, u_N, \xi_1, \ldots \xi_n) = 0 \\ &\vdots \\ w_N &= w_N(u_1, \ldots, u_N, \xi_1, \ldots \xi_n) = 0. \end{aligned} \tag{4.23}$$

Differentiating (4.23) with respect to the jth design variable, we find

$$\frac{\partial w}{\partial \xi_j} = \left(\frac{\partial w}{\partial \xi_j} \right)_u + J \frac{\partial u}{\partial \xi_j} = 0, \tag{4.24}$$

where

$$w = \left[w_1 \quad w_2 \quad \ldots \quad w_N \right]^T, \tag{4.25}$$

$$\frac{\partial w}{\partial \xi_j} = \left[\frac{\partial w_1}{\partial \xi_j} \quad \frac{\partial w_2}{\partial \xi_j} \quad \ldots \quad \frac{\partial w_N}{\partial \xi_j} \right]^T, \tag{4.26}$$

and

$$J \equiv \left(\frac{\partial w}{\partial u} \right)_\xi = \begin{bmatrix} \frac{\partial w_1}{\partial u_1} & \frac{\partial w_1}{\partial u_2} & \ldots & \frac{\partial w_1}{\partial u_N} \\ \frac{\partial w_2}{\partial u_1} & \frac{\partial w_2}{\partial u_2} & \ldots & \frac{\partial w_N}{\partial u_N} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial w_N}{\partial u_1} & \frac{\partial w_N}{\partial u_2} & \ldots & \frac{\partial w_N}{\partial u_N} \end{bmatrix}. \tag{4.27}$$

We can find $\partial u / \partial \xi_j$ by solving the linear system

$$J \frac{\partial u}{\partial \xi_j} = - \left(\frac{\partial w}{\partial \xi_j} \right)_u. \tag{4.28}$$

The $(\partial w/\partial \xi_j)_u$ term due to the appearance of the geometry in the governing equations, is evaluated numerically. The elements of the Jacobian, J, can be expressed as

$$\frac{\partial w_j}{\partial u_i} = \frac{1}{\Delta x}\left[\frac{\partial f_{j+1/2}}{\partial u_i} + \frac{\partial f_{j-1/2}}{\partial u_i}\right] + \frac{\partial g_j}{\partial u_i}, \tag{4.29}$$

where

$$\frac{\partial g_j}{\partial u_i} = \begin{cases} 0 & i = 1,\ldots,N; \quad i \neq j \\ \frac{A_{x_j}}{A_j}\frac{(\gamma-1)}{(\gamma+1)}\left(1+\frac{2h_o}{u_j^2}\right) & i = j \end{cases}. \tag{4.30}$$

The flux derivatives are dependent on the flow solver. For the Godunov scheme we have

$$\frac{\partial f_{j+1/2}}{\partial u_i} = \begin{cases} \frac{\partial f_{j+1}}{\partial u_i} & u_j, u_{j+1} < u_* \\ \frac{\partial f_j}{\partial u_i} & u_j, u_{j+1} > u_* \\ 0 & u_j < u_* < u_{j+1} \\ \frac{\partial f_j}{\partial u_i} & u_j > u_* > u_{j+1}; f_j > f_{j+1} \\ \frac{\partial f_{j+1}}{\partial u_i} & u_j > u_* > u_{j+1}; f_j < f_{j+1} \end{cases}. \tag{4.31}$$

where

$$\frac{\partial f_j}{\partial u_i} = \begin{cases} 0 & i-1,\ldots,N; \quad i \neq j \\ 1 - \frac{(\gamma-1)}{(\gamma+1)}\frac{2h_o}{u_j^2} & i = j \end{cases}. \tag{4.32}$$

For the artificial viscosity scheme we have

$$\frac{\partial f_{j+1/2}}{\partial u_i} = \begin{cases} 0 & i-1,\ldots,N; \quad i \neq j, j+1 \\ \frac{1}{2}\left[1 - \frac{(\gamma-1)}{(\gamma+1)}\frac{2h_o}{u_j^2} + \alpha\right] & i = j \\ \frac{1}{2}\left[1 - \frac{(\gamma-1)}{(\gamma+1)}\frac{2h_o}{u_j^2} - \alpha\right] & i = j+1 \end{cases}. \tag{4.33}$$

As in the continuous direct method, (4.28) represents one of n equations which must be solved to get the derivatives. Having solved (4.28) for $\partial u/\partial \xi_j$, the derivatives are computed from (4.21). For this problem, the Jacobian is tridiagonal so that the n solutions to (4.28) are inexpensive. In other design problems, the Jacobian may not be tridiagonal, but if the LU decomposition of J can be stored the n solutions of (4.28) can come at the expense of approximately one.

4.2 Adjoint Method

For the discrete adjoint method we define an augmented objective function

$$I^* = I + \lambda^T w, \tag{4.34}$$

where I is defined by (1.32), λ^T is a row vector of Lagrange multipliers, and w is the column vector of discretized governing equations. The sensitivity with respect to the jth design variable is

$$\frac{\partial I^*}{\partial \xi_j} = \frac{\partial I}{\partial \xi_j} + \lambda^T \frac{\partial w}{\partial \xi_j}. \tag{4.35}$$

Since w is zero in the entire domain, $\partial I/\partial \xi_j$ is equal to $\partial I^*/\partial \xi_j$. In the adjoint method we choose λ such that the $\partial u/\partial \xi_j$ terms will not appear in the sensitivity equation.

The first term on the right hand side has been expanded in the derivation of the discrete direct method (4.21). The derivative of the governing equation has also been expanded in equation (4.24). Substituting (4.21) and (4.24) into (4.35) yields

$$\frac{\partial I^*}{\partial \xi_j} = v^T \frac{\partial u}{\partial \xi_j} + \lambda^T \left[\left(\frac{\partial w}{\partial \xi_j} \right)_u + J \frac{\partial u}{\partial \xi_j} \right]. \tag{4.36}$$

We rearrange (4.36) to collect terms multiplying $\partial u/\partial \xi_j$ to get

$$\frac{\partial I^*}{\partial \xi_j} = \left(v^T + \lambda^T J \right) \frac{\partial u}{\partial \xi_j} + \lambda^T \left(\frac{\partial w}{\partial \xi_j} \right)_u. \tag{4.37}$$

The vector of Lagrange multipliers is arbitrary, but as in the continuous adjoint case, it may be chosen such that the sensitivity can be computed inexpensively. If λ is chosen such that

$$J^T \lambda = -v, \tag{4.38}$$

then the sensitivity is

$$\frac{\partial I}{\partial \xi_j} = \frac{\partial I^*}{\partial \xi_j} = \lambda^T \left\{ \frac{\partial w}{\partial \xi_j} \right\}_u. \tag{4.39}$$

The advantage this method has over the discrete direct method is evident with very large systems. Equation (4.38) has to be solved only once to calculate the Lagrange multipliers. The derivatives may then be computed inexpensively using (4.39) with different right-hand-sides, *i.e.* different $\partial w/\partial \xi_j$.

CHAPTER
FIVE

RESULTS: PART I

We now present design sensitivities and optimization results using the continuous and discrete approaches. For each approach we will compare direct and adjoint calculations with finite-difference calculations. The problem of a non-smooth objective function is handled through shock fitting with the continuous approach and with coordinate straining and shock penalty for the discrete approach. For this reason, continuous sensitivities cannot be compared with discrete sensitivities. A flow field of 64 grid points is used for this study.

An initial area distribution was chosen so that the computed shock is significantly far away from the target. Two case studies are presented in this section. In the first, we use a single design variable. In the second, we increase the complexity by using three design variables. Here the design variables represent area values at x-locations along the duct. A cubic spline fitted through these points describes the area completely. The initial area distribution was specifically selected so that both design cases describe exactly the same initial area. The starting design is shown in figure 5.1. Also shown in figure 5.1 is the intended optimize design. The optimized design, defined by (1.21), can also be described by a cubic spline passing through one or three design variables. Table 5.1 lists the x-locations, the initial value of the design variables, and the target values. During the design process the x-location of the design variables remains fixed as does the inlet and exit areas and slopes.

The duct is optimized by matching the velocity distribution calculated from the area to a target velocity distribution. The case studies investigate designs

performed using exact, Godunov and artificial viscosity flow solutions. These solvers are described in chapter 1 of this dissertation. Figures 5.2, 5.3, and 5.4 show the initial velocity distribution in comparison to the target for the three flow solving algorithms. To avoid any issues which may arise if the target velocity is not a solution to the governing equations, the target velocity is created using the area described by equation (1.21) and the flow solver used during the design process.

Optimizations are performed using the conjugate gradient method outlined in chapter 2. The step size taken in the descent direction is computed by bracketing the minimum in the search direction and minimizing a fitted quadratic polynomial within the limits of the bracket. Optimization stops when the objective function is smaller than 10^{-10} or when the change in the objective function from one iteration to the next is insignificant.

5.1 Designs Using the Continuous Approach

The resolution of the shock wave diminishes as we consider the exact, Godunov, and artificial viscosity solver. In the continuous approach the value of the velocity to the left and right of the shock is needed. The analytic solution identifies u^+ and u^- exactly by satisfying the jump condition (1.14) at the shock. The Godunov solver captures the shock generally over two or three grid cells allowing for estimates of u^+ and u^- to be extrapolated to the position where the velocity distribution crosses the sonic line. The artificial viscosity solver smears the shock to the degree where u^+ and u^- are poorly defined. For this reason, results using the artificial viscosity solver and the continuous approach are not presented.

Tables 5.2 through 5.5 list the sensitivities computed using the continuous direct and adjoint methods during the first design iteration for the univariate and the three design variable cases. Two finite-difference approximations are also included for the purpose of checking the accuracy of the design sensitivities. The

forward-difference requires less function evaluations than the more accurate central-difference, yet the sensitivity shows little effect of the difference approximation used in the calculations. A step size of 10^{-6} was used to compute the differences. Tables 5.2 and 5.3 show results computed using the exact flow solver while the results for the Godunov solver are in tables 5.4 and 5.5. Also listed in these tables are the values of the objective function after optimization is complete, and the number of iterations needed for convergence. In these cases the objective function is defined by (1.26). The convergence information gives an indication of the accuracy of the sensitivity derivatives.

In the cases using the exact flow solver, the derivative methods appear to give results in reasonable agreement with each other. Also the performance of the optimizer behaves similarly for all sensitivity methods. Table 5.6 lists the final values of the design variables. All methods converged to similar solutions, yet did not reach the intended design. The reason for this is apparently related to the nature of the objective function. The shock-fitted objective function shown in figure 1.10, expanded near the minimum reveals a cusp (figure 5.5) making convergence to the minimum difficult. A more robust optimizer should be able to find the minimum of the function.

Agreement between the sensitivity methods diminishes for optimization involving the Godunov solution. We feel that the reason for the disparity is related to the procedure used to locate the shock and fabricated values for u^+ and u^-. Recall, for numerical solutions to the governing differential equations, the shock position is found by interpolation between grid points. For the 64 grid point study, the shock is located to within 1.5%. In figure 5.6 the shock position is plotted over a small range of the design variable for the univariate case. The fabricated function used to define the shock position is wavy over the design space and can yield large

errors in the sensitivity calculation. Table 5.7 lists the shock sensitivities computed via finite-differences and the direct method for the analytic and Godunov solutions respectively.

There is better agreement with the exact solution as there is no ambiguity associated with locating the shock. The shock location and values of velocity across the shock can be found precisely. The agreement lessens in the Godunov derivatives due to the waviness described above. Note however, in table 5.7 the direct method computes sensitivities resembling those of the exact solution. This comes about because the jump relation (1.14) used to locate the shock in the exact solution is differentiated in the direct and adjoint formulations.

The situation is worsened by the extrapolation for the left and right velocity values at the shock. We have seen the waviness caused by the interpolation to get the shock position. The extrapolation compounds the issue not only because extrapolation is riskier than interpolation, but because the extrapolation is to the shock position, which itself is wavy. Figure 5.7 shows the waviness of the left and right values of velocity at the shock over a range of the design space for a univariate parameterization.

The final values of the design parameters for the Godunov case are listed in table 5.8. Together with the final values of the objective function in table 5.5, they indicate that there is still a difficulty in matching the target completely. The errors associated with locating the shock position and computing the left and right values of velocity at the shock introduce noise into the objective function and is amplified in the calculation of the derivatives. Thus with the Godunov flow solution, no sensitivity method stands out as being more accurate.

5.2 Designs Using the Discrete Approach

In the discrete approach, the sensitivities rely on the discretization of the gov-

erning equations. This approach is not applicable for the use with the exact flow solver. Results presented here use only the numerical flow solvers.

Tables 5.9 and 5.10 list the design sensitivities using the Godunov flow solver, while tables 5.11 and 5.12 list the design sensitivities using the artificial viscosity flow solver. The sensitivities are in excellent agreement and with the exception of the three design variable case using the Godunov flow solver, the optimizer performed similarly for each method.

Final design variables are shown in tables 5.13 and 5.14 for all discrete cases. One advantage of the coordinate-strained objective function over the shock-fitted function is clearly demonstrated with tables 5.13 and 5.14 and the final values of the objective function. That is, the intended design is recovered unlike the shock-fitted cases. The coordinate-straining transformation, while eliminating the stair-like structure, flattens out the function, while the shock penalty adds steepness. The important issue is that the cusp-like nature exhibited in the shock-fitted objective function is not present.

For most discrete cases presented, the optimization converged quickly. However, the optimization with finite-difference sensitivities for the case with three design variables and the Godunov solution showed a marked improvement over the analytic derivatives. To investigate this, the optimization was repeated using a 16 grid point flow domain in place of the 64 point domain. This experiment effectively smears the shock wave. In the 16 grid point case, all sensitivity calculations converged the design in 19 iterations. We conclude that the shock position, which becomes an increasingly wavy function of the design variable with the resolution of the flow solution, contaminates the derivative calculations. The direct and adjoint sensitivities are much more susceptible to the noise than the finite-difference derivatives since they directly compute shock position sensitivity. Further the noise in the shock position calculation begins to dominate near the minimum.

CHAPTER
SIX

CONCLUSIONS: PART I

In this part of the dissertation we have introduced an inverse design problem involving quasi-one-dimensional flow with a shock. We have demonstrated through graphic representations of the objective function that without proper consideration of the flow discontinuity, recovery of the target is difficult with derivative-based optimization. The objective function becomes non-smooth due to the interaction between the shock wave and the discretization of the minimization problem. As a result the sensitivities computed, although accurate for the non-smooth objective function, may not lead to the global minimum. We have indicated that when the shock is greatly smeared by the flow solver, the optimization difficulties become less severe as a result of the smoother objective function. However as the flow is better resolved, the difficulties will reappear. A robust sensitivity procedure must take into account the effect of shock waves.

We have presented two techniques to lessen the non-smooth nature of the objective function. One method involves shock fitting the objective function; the other involves a coordinate transformation which effectively removes the discontinuities from the objective function. We have shown these techniques to be more effective smoothers when the shock is precisely defined. For less sharp shock waves, such as those appearing in numerical solutions, we have demonstrated that these techniques only reduce the severity of the waviness. In the shock-fitted objective function, and where numerical flow solutions are used, the waviness is introduced through the interpolation routine to find the shock position, and the extrapolation to find the

left and right velocity values on either side of the shock. In the coordinate-strained and shock penalty objective function, the waviness enters only through the interpolation of the shock position as the left and right values of the velocity at the shock are not needed.

Several techniques were applied to the shock-fitted and the coordinate-strained objective functions. For the shock-fitted function we applied a direct and adjoint method in a continuous approach. The continuous approach differentiates the objective function and governing differential equation in their continuous form, *i.e.* before they are discretized for evaluation by the computer. The direct approach requires the solution to the flow equations plus solutions to n ordinary differential equations to evaluate the sensitivities. Here n is the dimension of the design space and the ordinary differential equations are called the direct equations. The adjoint method, by augmenting the objective function, requires the solution to the flow equation and only one solution to an ordinary differential equation. This is regardless of the dimension of the design space. Thus the direct problem grows with the dimension of the design space and the adjoint remains a constant size. In elementary design problems, such as this one, the direct and adjoint method are almost indistinguishable from a CPU stand point since the solutions to the boundary value problems are computationally inexpensive. In more complex problems the cost of solving the direct and adjoint equations will increase as analytic solutions may not be available. The advantage of the adjoint method will become more apparent from a CPU point of view in large design dimensional problems as only one adjoint equation needs to be solved for all the sensitivities. The direct method requires the solution of one direct equation for each design derivative.

The direct and adjoint methods were also applied to the coordinate-strained and shock penalty objective function. Here the techniques were applied in a discrete

sense. Before the sensitivity equations were developed, the flow field was discretized and the governing differential equation was replaced by a system of N algebraic equations. The objective function was also replaced by a discrete version. The direct method requires the solution to the system of flow equations, plus solutions to n systems of N linear algebraic equations. Unlike the flow equations which must be time marched, the solution to the direct system of equations involves a tridiagonal matrix. The adjoint, like its continuous counterpart, augments the objective function to produce savings in the sensitivity calculations. The adjoint method requires the flow solution plus only one solution to a system of N linear equations. Again the savings are not very apparent in this problem. However, as N increases or as the designs involve two- and three-dimensional flows the adjoint savings will be significant.

We have presented comparisons in the form of sensitivity calculations and optimization results. The analytic sensitivities were benchmarked against finite-difference calculations which are much more straight forward to compute, but also much more inefficient. In the continuous approach we have found that for a very sharp shock wave, the sensitivities computed by direct and adjoint methods match to within a few percent to those computed via finite-differences. When the shock is smeared, the direct and adjoint methods computed sensitivities that were more accurate than the finite-difference sensitivities. One reason for the discrepancies is the interpolation of the shock position and the extrapolation of u^+ and u^-. While these techniques are adequate for computing the shock position, u^+ and u^-, they are not adequate for computing the sensitivities. The direct and adjoint methods compute the sensitivities by direct differentiation of the analytic expressions, while the finite-differences do not. Thus the overall sensitivities are thought to be more accurate with the direct and adjoint method than with the finite-difference method.

The agreement between finite-difference sensitivities and the direct and adjoint sensitivities are much better in the discrete approach. The discrete approach does not require the extrapolations of u^+ and u^- and thus noise generated from this calculation does not enter in the derivative calculation.

One disadvantage of the shock-fitted objective function is that it exhibits a cusp shape near the minimum which may hinder convergence with simple optimizers. The coordinate-strained and shock penalty function exhibited better convergence properties.

Our research indicated that care must be taken with respect to shock waves (and other steep gradients) and that problems may become more severe as the flow solver better resolves the flow field. The discrete methodology used here was effective when incorporated with a modified objective function based on coordinate straining and shock penalty. A drawback was that this approach required the sensitivity of the shock position. The continuous methods discussed here were even more affected by the shock sensitivity and, for this approach, it was found that existing interpolation methods are adequate in determining the shock position but not for the sensitivities.

PART
TWO

RESPONSE SURFACE METHOD

"but those who hope in the Lord will
renew their strength. They will
soar on wings like eagles"

—Isaiah 40:31

CHAPTER
SEVEN

RESPONSE SURFACE OPTIMIZATION ALGORITHM

The response surface method is an alternative to derivative-based optimization schemes for noisy or non-smooth functions as it does not use derivative information. In this method the objective function and constraints are modeled by analytic surfaces which are then cheaply minimized using standard techniques. The purpose of the response surface is to capture the global features of the objective function while eliminating any small, high frequency noise which may exist. If the response surface is a good representation of the objective function, its minimum should lie near the optimum design. To complete the optimization we may proceed in one of two ways. One is to reduce the design space around the minimum and perform another response surface minimization. A series of response surfaces constructed over smaller and smaller regions of the design space can locate the minimum. The other is to switch to a derivative-based optimization using the minimum of the response surface as the starting point. The reason we can use derivatives to optimize near the minimum is because on a very fine scale the objective function may be smooth, and so derivatives become meaningful. It may take several response surface cycles to get close enough to the minimum for derivative-based optimization to be successful.

There are several advantages to response surface methods. The primary advantage is its overall robustness. The method does not rely on derivatives of the objective function which can be corrupt in the presence of computational or experimental noise. Response surfaces are smooth, analytic functions whose derivatives

can be computed accurately and quickly. Noise generated in CFD and applied computational aerodynamics codes can generate errors in the objective function. For example, noise can be generated by changes in grids, discontinuities in flux limiters, or changes in the stopping criteria for iterative schemes. Many times noise will prevent derivative-based optimization from locating the minimum.

Another advantage is that the method probes a large region of the design space. Derivative-based procedures follow a path through the design space from the initial design to the final design. No function information from outside the path is ever used in the procedure. As a result, unless several different starting designs are investigated, the derivative-based optimization may lead to a local minimization instead of a global one. The response surface is modeled by sampling the design space over a relatively large region reducing the chance of converging to a local minimum.

A third advantage of the response surface method is that it is readily parallelizable. The construction of the response surface requires many independent analyses over the region of the feasible design space. When the analyses are computer generated, each data point can be analyzed on a separate processor of a multiprocessor machine. We will show that in this way, the data needed to construct the response surface can be gathered in the time it takes to make one analysis on a comparable single processor machine. This advantage is valuable when the analyses are expensive as they are in aerodynamic shape designs involving complicated flow simulations.

As with all optimization techniques, there are disadvantages. One stems from what is termed the "curse of dimensionality." As the dimension of the problem increases, the number of coefficients in the response surface rises rapidly as does the number of function evaluations needed to construct the surface. Further, the

least-squares problem which is solved to determine the coefficients of the response surface can become ill-conditioned in high dimensions of the design space. Problems solved by response surfaces in this work are parameterized in low dimensions. The solution to higher dimensional problems is an area for future work.

The purpose of the response surface is to capture the general shape of the objective function. Without experience or prior knowledge of the behavior of the objective function, it may be difficult to adequately model the function and locate the general vicinity of the optimum. Thus another disadvantage of this method is that a poor response surface model may slow or prevent convergence. Yet in small enough regions, most functions can often be well represented by quadratics.

The response surface method is comprised of one or more cycles. Each cycle contains the following steps:

- determine the region of the design space to search for the minimum,
- construct a response surface in this region,
- minimize the response surface,
- check for convergence.

First, we must determine an initial region to start our search. This is an advantage over derivative-based routines which generally require starting the optimization from a point. Here, instead of guessing the location of the point, we guess an entire region where the minimum is most likely to occur. However, some experience is needed to choose a region whose size is not too large or too small. An initial region which is too large is difficult to model accurately with a single response surface. A region which is too small has the chance of not including the minimum and will result in wasted cycles translating through the design space.

The construction of the response surface requires several steps. It requires determining the response function type, the number of points to use to construct

the surface, the location of the points in the design space to probe the objective function and constraints, and solving the least-squares problem for the coefficients to the response surface. These steps are expounded in the next chapter.

Minimization of the response surface is simple. The response surfaces modeling the function and constraints are smooth analytic functions that are cheap to evaluate. The minimization can be done using any method, however care must be taken to avoid local minima. In this work, the response surfaces are minimized using Schittkowski's sequential quadratic programming (SQP) code.[71] When there are no constraints, the SQP program reduces to a quasi-Newton method with BFGS Hessian updates. Minimizations are performed from several starting points to insure that the derivative-based optimization locates the absolute minimum.

Convergence of the response surface method can be detected a number of ways. For the inverse design problems, the minimum is found when the objective function is zero. For these designs, the stopping criterion detects when the objective functions reaches zero to within some small tolerance. For direct designs we observe the trends in the values of the design variables and objective function. When the values stabilize, we consider the design converged.

The heart of the response surface method rests in the way the region containing the surface translates and reduces in the design space. Response surface optimization reduces the region around the global minimum. As the design space shrinks, the variation of the objective function and constraints decreases and can be better modeled by the response surfaces. The dynamics and size reduction of the region is based on the minimization from the previous response surface. The design space region for the $i+1$ cycle is centered about the minimum of the response surface of the ith cycle. If the ith response surface minimum is against the boundary of the ξ_j design variable, then the design space is not reduced in the j direction. On the

other hand, if the minimum is not against the boundary, the region spanning the j direction is reduced by a factor of 4. The 75% reduction is an arbitrarily chosen factor and seems to work well in the cases we explored.

The reduction and translation of the response surface region is clearly demonstrated through a simple example. For the purpose of illustration we minimize the function

$$1 - 2.5x_2 + 2x_2^2 - 2.5x_1 - 6x_1x_2 + 10x_1x_2^2 + 3x_1^2 + 10x_1^2x_2 - 12x_1^2x_2^2, \tag{7.1}$$

where we allow

$$0 \leq x_{1,2} \leq 1. \tag{7.2}$$

The function can easily be analyzed to find that the minimum lies at $x_1 = 0.3184$, $x_2 = 0.4281$. The response surface we choose for this example is of the form

$$p = c_1 + c_2x_1 + c_3x_2 + c_4x_1^2 + c_5x_2^2 + c_6x_1x_2. \tag{7.3}$$

Note that the response surface will not be able to match the function exactly. The initial response surface region is chosen to be

$$0.5 \leq x_{1,2} \leq 1.0. \tag{7.4}$$

This region is purposely chosen so that the minimum lies outside its boundaries. Typically one would like to choose the initial region to contain the minimum, if at all possible, to expedite the optimization. However, the choice of initial region demonstrates the method's ability to find the minimum even when the minimum lies outside of it. The response surface is fitted to the function with nine points using methods to be discussed in the next chapter. The minimum, after one response surface cycle is approximated to be $x_1 = 0.5$, $x_2 = 0.5$. Since this point is on both the x_1 and x_2 boundary, no reduction of the response surface region is made for the

next iteration. Instead, the region is translated around $x_1 = 0.5, x_2 = 0.5$. This leads to the new region

$$0.25 \leq x_{1,2} \leq 0.75. \tag{7.5}$$

A new response surface is fitted in this region. Minimization of this surface leads to the point $x_1 = 0.3037$, $x_2 = 0.4251$. Here, the minimum does not lie on the boundary of the response surface region in either the x_1 or x_2 directions so that the new region will span a distance of 0.125 (0.25 times the previous span) in both directions. The new region is also translated to center around the minimum so that

$$0.2412 \leq x_1 \leq 0.3662,$$

$$0.3626 \leq x_2 \leq 0.4876. \tag{7.6}$$

Contours of the function and a map of the movements of the response surface region are shown in figure 7.1 for 4 response surface cycles. The circles in the figure represent where the the function is evaluated for the construction of the first three surfaces. The minima of the response surfaces are represented by the $\times$'s. After 4 response surface cycles, the function minimizes to $x_1 = 0.3184, x_2 = 0.4281$.

CHAPTER
EIGHT

RECIPE FOR RESPONSE SURFACE CONSTRUCTION

In the previous chapter we explained the general algorithm for response surface optimization. We presented a simple example that demonstrated the effectiveness of the response surface region in moving towards and surrounding the minimum. We now discuss the important details to constructing the response surface. There are four major steps involved in its construction. The steps are

- determine the function type for the response surface,
- determine the number of function evaluations to be used as the database for fitting the response surface,
- determine the location of the points used for fitting the response surface,
- fit the function by solving a least-squares problem.

The following chapter sections provide more details.

8.1 Selecting Response Surface Type

This section provides some insight into the selection of the shape of the response surface. The selection is dependent on the problem and no scientific rules are available for this. A wise selection of the shape of the response surface requires some knowledge of how the objective function behaves. Generally this is not known *a priori*, but with experience the selection will become more apparent. In this

work we experiment with polynomials, specifically quadratics and quadratic tensor products. The quadratic polynomial is of the form

$$\begin{aligned} P =& c_{11}\xi_1^2 + c_{12}\xi_1\xi_2 + c_{13}\xi_1\xi_3 + \ldots + c_{22}\xi_2^2 + c_{23}\xi_2\xi_3 + \ldots \\ &+ c_1\xi_1 + c_2\xi_2 + c_3\xi_3 + \ldots + c. \end{aligned} \tag{8.1}$$

A quadratic tensor product is of the form

$$P = (c_1\xi_1^2 + c_2\xi_1 + c_3)(c_4\xi_2^2 + c_5\xi_2 + c_6)\ldots \tag{8.2}$$

The coefficients are found by evaluating the objective function at several points and solving a least-squares problem. For the family of polynomials represented by (8.1) the least-squares problem is linear, however for (8.2) the coefficients require a solution to a difficult non-linear system. To avoid this, we multiply out the terms in (8.2) and solve the larger, yet linear least-squares problem. For example in two design variables ($n = 2$), we fit the polynomial

$$P = c_1\xi_1^2\xi_2^2 + c_2\xi_1^2\xi_2 + c_3\xi_1^2 + c_4\xi_1\xi_2^2 + c_5\xi_1\xi_2 + c_6\xi_1 + c_7\xi_2^2 + c_8\xi_2 + c_9. \tag{8.3}$$

Thus we solve for 9 coefficients instead of 8. In higher dimensions we can expect to solve for many more coefficients. In general, the number of coefficients rises as 3^n when the quadratic tensor product is expanded. Since the quadratic tensor product requires 3^n coefficients and hence many function evaluations to solve the least-squares problem, we use this polynomial only in low-dimensional problems and when the objective function is cheap to compute. In comparison, the quadratic polynomial has $(n+1)(n+2)/2$ coefficients and thus rises as n^2.

Other types of response surfaces have been experimented with. In structures, Toropov[38] used intrinsically linear functions in the coefficients. These are functions which are non-linear but can become linear through simple transformations. Toropov suggests the multiplicative function

$$P = c_0\xi_1^{c_1}\xi_2^{c_2}\ldots\xi_n^{c_n}, \tag{8.4}$$

with the transformation

$$\ln(P) = \ln c_0 + \sum_{i=1}^{n} c_i \ln \xi_i, \tag{8.5}$$

and the power function

$$P = \left(c_0 + \sum_{i=1}^{n} c_i \xi_i \right)^{\alpha}, \tag{8.6}$$

with the transformation

$$(P)^{1\alpha} = c_0 + \sum_{i=1}^{n} c_i \xi_i. \tag{8.7}$$

Certainly non-linear functions may be used to represent the response surfaces such as (8.2) in product form. In such cases the least-squares estimation for the coefficients would have to be solved using nonlinear programming. In the work presented here we will be only considering quadratic and quadratic tensor product response surfaces of the form (8.1) and (8.2).

8.2 Number of Function Evaluations for Fit

When determining the number of function evaluations which should be used to generate the response surface, we must consider the following three objectives. First, we must sample the design space enough times so that the response surface captures the major features of the objective function. Second, we must keep the condition number of the least-squares matrix down to an acceptable level so that we may accurately solve for the coefficients of the response surface polynomial, and third, we must minimize the amount of function calls to the analysis code to keep the computational cost down. The first two objectives are opposed to the third. Assurance that we capture all the features of the design space implies we make as many function evaluations as possible. Also generally, the condition number of the least-squares matrix increases as the number of points in the fit decreases, thus we

can get better conditioned matrices by gathering more data. Yet the many function evaluations which help us meet the first two objectives can be computationally intensive.

Although techniques for solving the least-squares problem with high condition numbers are available, the condition number can become unmanageably large especially for problems with a large number of coefficients. A polynomial with a large number of coefficients is characteristic of design problems involving a large number of design variables. It has been our experience that matrices with high condition numbers can lead to ill-conditioned solutions for the coefficients which often provide accurate models of the function, but poor estimates of the derivatives. Thus minimization of the response surface region by derivative-based optimization techniques becomes difficult.

The solution to the number of points to use is not general, as it depends on the affordability of evaluating the objective function and the shape of the design space. With careful selection of points in the design space it is possible to decrease the condition number of the least-squares matrix, thus requiring less points. It has been suggested that the number of points used in the fit should exceed the number of coefficients in the polynomial by 20% to 50%.[72] In this work, we perform our experiments using roughly 1.5 times the number of coefficients.

8.3 Location of Points to Construct Surface

Once the choice of curves and the number of points has been selected, we must decide where to evaluate the objective function. The choice of positions to gather data can have a profound effect on the fidelity of the response surface to model the objective function. Consider, for example, fitting a line through two points. Choosing to generate the data with points close together can yield large errors in

the fit with only small errors in the evaluation of the function. This fact is shown clearly in figure 8.1. Choosing to gather data at the ends of the domain is a much wiser choice.

The most general point selection set is the factorial set. With factorial sets, each axis of the design space is divided into k discrete levels. Every combination of the levels in each design direction makes up the points in the factorial set. For example, five levels in three-dimensional space (known as the 5^3 factorial set) leads to 125 points for the response surface. Factorial sets are attractive because they provide a uniform sampling over the region of interest and generally lead to well-conditioned least-squares matrices. However, they also lead to sets with large numbers of points, especially in high-dimensional space.

Alternatives to factorial sets, studied by Carpenter[72] and Giunta *et al.*[45] are central composite sets and optimality sets. Central composite sets can be viable solutions in low dimensions (< 6). In higher dimensions these sets, like factorial sets, yield many points and evaluating the objective function at all the design points can be prohibitively expensive. Another drawback of the central composite sets is that the points may exist outside the feasible domain. Often, the design space has an irregular shape due to constraints and the points which make the central composite sets may violate the constraints. Central composite sets are intended for unconstrained domains.

Optimality sets are sets which, based on statistics, yield points that give the best fit to the objective function for a specified number of points. Because, generally, the calculation of the objective function involves numerical errors, there will be errors in the computation of the coefficients of the response surface and errors in the response surface itself. A measure of the error is the variance. The variance of the coefficients are the diagonal terms of the variance-covariance matrix given by[72]

$$cov(c) = \sigma^2 (A^T A)^{-1}, \tag{8.8}$$

where

$$\sigma^2 = \frac{1}{n}\sum_{i=1}^{n}(I_i - \bar{I}_i)^2, \tag{8.9}$$

and A is the least-squares matrix. Details of the A matrix are given in section 8.4. In (8.8) and (8.9), σ^2 is the variance of the objective function, I_i is the value of the objective function at the ξ_i point, and $\bar{I}_i$ is the average value of I_i over n observations. The variance of the response surface, P, evaluated at ξ_i, is given by[72]

$$var(P_i) = \sigma^2 A_i^T (A^T A)^{-1} A_i, \tag{8.10}$$

where A_i is a row of the A matrix. To keep the variance of both the coefficients and the response surface low it is advantageous to minimize the matrix $(A^T A)^{-1}$. Minimization of a matrix is not a well defined concept and this leads to several optimality criteria. These are discussed in reference 72. One, which is consistent with picking the end points in the simple example of figure 8.1, is the D-optimal criteria.

The D-optimal criterion minimizes $(A^T A)^{-1}$ by minimizing the determinant. This is equivalent to minimizing the product of eigenvalues of $(A^T A)^{-1}$ or maximizing $|A^T A|$. If m is the number of points used to construct the response surface, and l is the number of points in the mesh to describe the domain of independent variables then the D-optimal criterion is satisfied when the set of m points chosen for the least-squares problem from the pool of l points is such that $|A^T A|$ is a maximum. For example, in a two-dimensional domain for which we wish to model the objective function using the response surface, $P = c_o + c_1\xi_1 + c_2\xi_2 + c_3\xi_1^2 + c_4\xi_2^2 + c_5\xi_1\xi_2$, the A matrix has the form

$$A = \begin{pmatrix} 1 & \xi_{1,1} & \xi_{2,1} & \xi_{1,1}^2 & \xi_{2,1}^2 & \xi_{1,1}\xi_{2,1} \\ 1 & \xi_{1,2} & \xi_{2,2} & \xi_{1,2}^2 & \xi_{2,2}^2 & \xi_{1,2}\xi_{2,2} \\ & & & \vdots & & \\ 1 & \xi_{1,9} & \xi_{2,9} & \xi_{1,9}^2 & \xi_{2,9}^2 & \xi_{1,9}\xi_{2,9} \end{pmatrix}, \tag{8.11}$$

where the first subscript in the ξ variable represents the design coordinate, and the second represents the point; there are nine points being used to solve the least-squares problem. If we discretize the domain into 36 points as shown in figure 8.2, the nine blackened points are those which maximize $|A^TA|$. The problem of choosing the best m points from l possible points to fit a curve reduces to choosing the set of m points which maximizes $|A^TA|$. To reiterate, the criterion is appealing for choosing the points for the least-squares problem because of the following two properties[73]

- the set of points that maximizes $|A^TA|$ is also the set of points that minimizes the maximum variance of any predicted value of the objective function,
- the set of points that maximizes $|A^TA|$ is also the set of points that minimizes the variance of the coefficients,

The D-optimal criterion also has the property that D-optimal points are invariant to changes in scale of the domain of independent variables. This means the D-optimal points for a square domain are proportionally in the same position for a rectangular domain. In addition by forcing the determinant of A^TA to be large, we are making the least-squares problem well conditioned. A more detailed discussion of the D-optimality criterion is presented by Box and Draper.[73]

There are other criteria used for minimizing $(A^TA)^{-1}$. These include A-optimality which minimizes the trace of $(A^TA)^{-1}$, and E-optimality which minimizes the largest eigenvalue of $(A^TA)^{-1}$. A discussion of these and other criteria is presented in reference 72. The conclusion of the study indicated that surfaces constructed using D-optimal points showed the best fidelity to the actual function.

As an example of the usefulness of the D-optimal criterion, consider the function

$$y = \sin(\xi_1\pi)\sin(\xi_2\pi), \tag{8.12}$$

defined on the region from

$$0 \leq \xi_{1,2} \leq 1. \tag{8.13}$$

We wish to approximate this function with a biquadratic tensor product of the form of (8.3) using 9 points, the minimum we can use. We choose to discretize the domain in a rectangular 5×5 grid as in figure 8.2 (5^2 factorial design). To obtain the D-optimal points we must find the 9 points out of 25 which have the highest $|A^T A|$. A brute force method is to check every combination of 9 points from 25. This leads to 2.0×10^6 combinations according to the rule

$$\binom{l}{m} = \frac{l!}{m!(l-m)!}. \tag{8.14}$$

This is what is performed here. The result is shown in figure 8.3. The empty circles represent the points which are used to describe the domain over which the response surface is to be defined. The blackened points are the D-optimal points. In this case there is only one set of points in the 2.0×10^6 possibilities that is D-optimal. Generally, this does not have to be the case. If other sets of points have $|A^T A|$ of equal value then multiple sets of points satisfy the D-optimal criterion. Figure 8.4 shows four other possibilities of nine points which may have been chosen for the response surface construction. These are not D-optimal points but serve as a basis for comparing the quality of fit of D-optimal to non-D-optimal points. Results of a comparison of the fit to the transcendental function, (8.12), are compiled in table 8.1. An error parameter defined by

$$V = \frac{\sqrt{\frac{1}{N}\sum_{i=1}^{N}(I_i - P_i)^2}}{\frac{1}{N}\sum_{i=1}^{N} I_i}, \tag{8.15}$$

is used to measure the quality of fit. In (8.15), I is the function value, P is the polynomial fit, and the summation is taken over $10,201$ evenly distributed points in the domain.

The general trend is that the higher the value of $|A^T A|$ the better the fit, although there are clearly exceptions to this rule. The D-optimal set of points has the lowest error as defined by (8.15). This example also shows that points placed along the edges of the domain tend to increase the fidelity between the response surface and the function. Yet placing all the points along the edge would have an obvious disadvantage in modeling the interior of the domain. D-optimal sets do tend to place many points along the edge of the domain, but usually have interior points as well.

The problem of finding the m points to satisfy D-optimality requires considering the $\binom{l}{m} = l!/(m!(l-m)!)$ combinations of m points from the set of l candidate points. A small problem in two design variables may be to pick 25 points from a selection of 121 (discretizing the domain into 10 sections in both directions gives an 11x11 mesh). This leads to a total of 5.26×10^{25} possible combinations, one or more of which are D-optimal. Clearly, even for this small problem, checking all combinations is infeasible. Standard methods for maximization can run into three problems. First, the number of variables can be quite large. Each point in the discretized domain represents a variable in the maximization problem. Second, the maximization is of an integer type. Each of the points can either be included or not. Finally, $|A^T A|$ may have local maxima. Several researchers have developed algorithms to search for D-optimality without checking every combination.[74,75,76] Perhaps the most popular algorithm is Mitchell's DETMAX code.[76] In this work we employ a genetic algorithm (GA) which we feel is an improvement over DETMAX. Details of the GA used to find D-optimality are given in Chapter 9.

8.4 The Least-squares Problem

The least-squares problem, as formulated in reference 78, can be described as follows: We are given a set of data points in n-dimensional space. We want to write

these points in a subspace of $\Re^n$, but we cannot since the system is over determined. The least-squares solution to the over determined system is to find the set of points in the subspace of $\Re^n$ that most closely matches the data. Say for example, in two-dimensional space we are given three points which we wish to describe with a plane passing through the origin,

$$P = c_0 x + c_1 y. \tag{8.16}$$

To determine the coefficients, c_0 and c_1, we would need to solve the system of equations

$$\begin{bmatrix} x_1 & y_1 \\ x_2 & y_2 \\ x_3 & y_3 \end{bmatrix} \begin{Bmatrix} c_0 \\ c_1 \end{Bmatrix} = \begin{Bmatrix} z_1 \\ z_2 \\ z_3 \end{Bmatrix}, \tag{8.17}$$

which, for the general case would be inconsistent (three equations for two unknowns). Equation (8.17) can be more concisely represented as $Ac = z$. The least-squares solution to the over determined system is to minimize the difference between the polynomial and the function. The problem can be written as

$$\min \|Ac - z\|_2 . \tag{8.18}$$

To minimize (8.18), the error vector, $Ac - z$, must be perpendicular to the column space of the matrix A. This means any linear combination of the columns of A must be perpendicular to $Ac - z$. This is illustrated in figure 8.5 and represented as

$$(Ad)^T(Ac - z) = 0, \tag{8.19}$$

where d represents the coefficients of the linear combination of column vectors of A. Rearranging we can write that

$$d^T(A^T Ac - A^T z) = 0. \tag{8.20}$$

If (8.20) is true for every d vector, then

$$A^T Ac - A^T z = 0, \tag{8.21}$$

and the least-squares solution to a set of linear, inconsistent equations also satisfies

$$A^T Ac = A^T z. \tag{8.22}$$

If the columns of A are linearly independent, then the matrix can be factored into an orthogonal matrix, Q, and an upper triangular matrix, R. In this way, $A^T A = R^T Q^T QR$. Any orthogonal matrix has the property that $Q^T = Q^{-1}$ so that $Q^T Q = I$. Thus (8.22) can be simplified to

$$Rc = Q^T z \tag{8.23}$$

The vector of coefficients describing the subspace can be easily obtained by backwards substitution.

CHAPTER
NINE

GENETIC RECOMBINATION ALGORITHM FOR MULTIPLE POINT SELECTION

GRAMPS, or Genetic Recombination Algorithm for Multiple Point Selection, is a genetic algorithm (GA) code whose purpose is to assist in the construction of the response surface by seeking the D-optimal points for sampling the function. This chapter describes the details of the genetic algorithm. Appendix A is a user's guide to the code. A source listing of the code is located in Appendix B.

The GA employed here is roughly modeled after the algorithm described in the paper by Furuya and Haftka.[78] Unlike some conventional optimization techniques which work in the neighborhood of a design point, the GA works by investigating several designs over the entire design space. The group of designs sampled by the GA makes up the population. Each design in the population is evaluated and ranked according to some cost function. More appealing designs have a higher rank. The next step in the GA is the breeding process where designs are selected for parenting child designs. The probability of a design being selected to be a parent is weighted according to the ranking so that children are made from the best designs. A child is created by combining parts from two parents. A probability of mutation is introduced in the GA to allow for the inclusion of certain aspects of the design not present in the parent generation. Mutation also guards against all designs of the population becoming the same. A new generation is formed from the child designs and the best design of the previous generation, with the new generation having the same population size as the previous. The child generation then becomes the new

parent generation and the process continues over many generations. The design with the best cost function in the end is considered the optimum design.

The specifics of the GA as applied to finding D-optimality are as follows. To start the algorithm, an initial population of candidate designs is created. Each candidate is formed by randomly selecting m distinct points from the set of points describing the region. These candidate designs are then ranked among each other with the best design having the highest value of $|A^TA|$. The cost function, $|A^TA|$ is efficiently computed by performing a QR factorization on A. In this way the determinant is calculated by multiplying the square of the diagonal elements of the R matrix

$$|A^TA| = |R|^2. \tag{9.1}$$

The population is selected for breeding based on a fitness parameter, f, which is related to the rank of the design by

$$f = b + 1 - r, \tag{9.2}$$

where b is the number of designs in the population and r is the rank. In this way, the design with a #1 ranking will have the highest fitness. The probability of the rth ranked design being selected as a parent is defined as

$$p_r = \frac{2f}{b(b+1)}. \tag{9.3}$$

The selection process for parenting is completed by generating a uniformly distributed random number, x, between zero and one, and selecting the rth ranked design satisfying

$$P_r \leq x \leq P_{r+1}, \tag{9.4}$$

where,

$$P_r = \sum_{i=1}^{r-1} p_i. \tag{9.5}$$

This is called the ranked-based fitness technique for selection. For a population of 10 designs the probability of each design being selected for parenting is shown graphically in figure 9.1.

After the parents have been selected, the child design is made by combining the two parents. This is done by describing the designs with a string of genes. Typically, the genes are coded versions of the design variables and distinguish one design from another. A random integer, j, ranging between 1 and the number of genes minus one is generated. The child design is made using the first j genes of parent 1; the remaining genes come from parent 2. Once the child is generated, it goes through a mutation process. In the work presented, each gene has a 15% chance of mutating. If a gene is selected for mutation, the gene is replaced with a gene coming randomly from the set of allowable values for that gene. By this method, baring the chance of mutation, the child design will have at least one gene from each parent. Finally, the child is checked against having duplicate points. Since the set of points will be used to generate surfaces, it is not desirable to have repeated points. If the design has duplicate points, the child is destroyed and the parent selection process is repeated. The child breeding process is shown schematically in figure 9.2. In this figure the genes are represented by integer values. In this case the fifth gene of the string was selected for mutation.

For a general-shaped space, the string of genes is comprised of the m points selected for making the response surface. Each point used to describe the space is assigned a number so that the genetic string is a series of integers as in the example of figure 9.2. In a genetic string no numbers may be repeated.

If the space is a hypercube, that is we can describe the space by only supplying a lower and upper limit in all dimensions, we use an alternate description for the genetic string. In such a case, the string is comprised of the coordinates of the points

chosen to make the response surface. For example, in two-dimensional rectangular space described by 25 points, the genetic string of a particular design containing 9 points may be

$$x_2\ y_2,\ x_4\ y_3,\ x_3\ y_2,\ x_5\ y_1,\ x_5\ y_5,\ x_1\ y_1,\ x_1\ y_5,\ x_4\ y_5,\ x_2\ y_4.$$

In the above example, the coordinates of some genes are repeated and such a case is allowed. Duplicate points in the design are not allowed and thus each x and y pair cannot be repeated. The x and y pairs are separated by commas in the above genetic string representation. Designs described in this manner tend to find the D-optimal set of points faster since there is more freedom for the design to change form throughout the generations. This representation cannot be used for the general design spaces since the solution from the GA may lead to points defined outside the allowable region.

After $b-1$ children are created, the parent generation is replaced. Only the best parent design is retained to insure that the best design survives throughout the generations. After a given number of generations have been bred, or a given number of $|A^TA|$ have been computed, the candidate with the highest $|A^TA|$ is used for forming the response surface. There is no guarantee that the output of the GA is truly D-optimal. However, known D-optimal sets of points for test cases have been recovered.

To show this, we repeat the example done in chapter 2 where we seek nine D-optimal points for the construction of a two-dimensional response surface using the quadratic given by (8.3). We set the GA to run with a population of 5 using the coordinates of the points for the genetic string representation. The GA ran for 500 generations, and the history of $\max|A^TA|$ for the population is shown in figure 9.3. The D-optimal set of points was recovered after 243 generations. In the first

generation, $|A^TA|$ was computed 5 times, once for each design in the population. In the ensuing generations, $|A^TA|$ was computed 4 time per generation, once for each child. Thus $|A^TA|$ was computed a total of 2001 times; a huge savings compared to the $2.0x10^6$ times required to check every combination of 9 points from 25.

As another example, consider the region described by the 38 points shown in figure 9.4. Here the region is not a rectangle, so we must represent each design using points for the genetic string. Here we choose to represent the response surface as

$$P = c_0 + c_1x + c_2x^2 + c_3y + c_4xy + c_5x^2y, \tag{9.6}$$

anticipating a greater variation in the x-direction than in the y. The response surface requires a minimum of 6 points to define it, here we use 9. There are $1.63x10^8$ possible combinations of 9 points from 38. The genetic algorithm ran for 1500 generations using a population of 5 designs. This required 6001 computations of $|A^TA|$. The history of $\max|A^TA|$ is shown in figure 9.5. We speculate that the D-optimal set of points (the blackened dots of figure 9.5) was reached in the $869th$ generation, although we cannot tell with absolute certainty that this is truly D-optimal without running all $1.63x10^8$ possible combinations. Due to the random nature of the GA, most searches for D-optimal sets should be repeated several times with various random number seeds. The repeatability of the GA will increase the confidence that the points selected are D-optimal or nearly D-optimal.

CHAPTER
TEN

DESIGN PROBLEMS

The response surface technique will be demonstrated using three problems. The first two are of the inverse design type; the last is a direct design problem. The first problem is the inverse design of a duct using quasi-one-dimensional flow theory. It is the problem discussed in Part 1 and results of the response surface will be compared to the derivative-based methods discussed earlier. The next design problem involves the matching of a pressure profile over a bump in a channel of transonic flow. The third problem involves maximizing lift on a two-dimensional airfoil with constraints on drag and area. The bump and airfoil problems are discussed in the next subsections.

10.1 Inverse Design of a Bump in Transonic Channel Flow

For this problem we wish to describe the shape of a bump located inside a channel so that we may recover a given pressure distribution. This problem, though academic, serves for two reasons. First, it demonstrates the ease at which the response surface technique can be applied to two-dimensional problems. The methods in Part 1 require a fair amount of calculus in preparing the analytic derivatives in the extension from one-dimension to two. No part of the algorithm changes in the response surface method except for the installation of an existing two-dimensional flow solver. The second reason for the problem is that it acts as a stepping stone to the airfoil problem to be discussed in the next section.

The geometry of the channel is shown in figure 10.1. It has a length of 5 units and a height of two units. The bump is centered along the bottom of the channel

and has a length of one. The bump is parameterized with the four shape functions shown in figure 10.2. These shape functions are generated by fitting a B-spline through the points listed in table 10.1. The points ensure that the bump will begin and end at the points (0,0) and (1,0), *i.e.* no gaps on the channel floor. The shape of the bump is constructed from a weighted sum of the shape functions

$$Y = \sum_{i=1}^{4} \xi_i y_i, \tag{10.1}$$

where y_i are the shape functions and the weights, ξ_i, are the design variables.

The flow through the channel is a solution to the Euler equations written for a perfect gas. In two-dimensions, the conservative form of the governing equations can be expressed in Cartesian coordinates as

$$\frac{\partial Q}{\partial t} + \frac{\partial F}{\partial x} + \frac{\partial G}{\partial y} = 0, \tag{10.2}$$

where

$$Q = \begin{Bmatrix} \rho \\ \rho u \\ \rho v \\ \rho e_o \end{Bmatrix}, \quad F = \begin{Bmatrix} \rho u \\ \rho u^2 + p \\ \rho uv \\ (\rho e_o + p)u \end{Bmatrix}, \quad G = \begin{Bmatrix} \rho v \\ \rho uv \\ \rho v^2 + p \\ (\rho e_o + p)v \end{Bmatrix}. \tag{10.3}$$

The Euler equations are solved using ErICA, a two-dimensional finite-volume, upwind, implicit solver. The author has developed ErICA at the Virginia Tech Aerospace Engineering Computer Laboratory. Details of the code are supplied in Appendix C. The Euler equations are solved for a Mach 0.80 flow on an 81x31 point grid, with 41 points on the bump. A typical grid is shown in figure 10.3. Mesh sequencing and multigrid are used to accelerate convergence.

The objective function for this problem is defined as

$$I(\xi) = \frac{1}{2} \sum_{i=1}^{41} (\hat{C}_p - C_p)_i^2, \tag{10.4}$$

where $\hat{C}_p$ is the target pressure coefficient distribution, C_p is the design presure distribution, and the summation is taken over the grid points on the surface of the bump. The pressure coefficient is defined in the usual way, $C_p = 2(p_\infty - p)/\rho V_\infty^2$. The target profile is created by solving the Euler equations over a bump described by

$$\hat{y} = \sqrt{1.3 - (x - 0.5)^2} - 1.2. \qquad 0 \leq x \leq 1 \tag{10.5}$$

Pressure contours of the target solution are shown in figure 10.4.

The minimization of the objective function is subject to constraints on the area

$$0.03 \leq A \leq 0.08. \tag{10.6}$$

The area constraints keep the shape of the bump in a section of design space where flow solutions do not suffer from convergence difficulties.

The objective function is non-smooth like the problem involving the one-dimensional transonic flow through a duct. The adverse effects of the interaction between the shock wave and the discretization of the domain can be visualized with a one-dimensional cut through the design space as shown in figure 10.5. Here the shape of the bump is varied linearly according to

$$\begin{aligned}
\xi_1 &= 0.3406 + \alpha(0.2543 - 0.3406), \\
\xi_2 &= 0.4640 + \alpha(0.5463 - 0.4640), \\
\xi_3 &= 0.7500 + \alpha(0.5463 - 0.7500), \\
\xi_4 &= 0.1750 + \alpha(0.2543 - 0.1750).
\end{aligned} \tag{10.7}$$

10.2 Transonic Airfoil Design

Response surface technology is also useful for more practical applications such as airfoil design. In this section, we present the model problem of Vanderplaats *et al.*[79] as implemented by Joh *et al.*[61]

The airfoil shape, of unit chord, is described by a weighted sum of six shape functions

$$Y = \sum_{i=1}^{6} \xi_i y_i(x/c). \tag{10.8}$$

Four of the shape functions, y_1-y_4, are pre-existing airfoils, namely NACA 2412, NACA 64_1-412, NACA 65_2-415 and NACA 64_2-A215. The values for y_1-y_4 may be found in reference 80. The remaining two shape functions are

$$y_5 = \begin{cases} x/c, & \text{on upper surface} \\ 0, & \text{on lower surface} \end{cases}, \tag{10.9}$$

$$y_6 = \begin{cases} 0, & \text{on upper surface} \\ -x/c, & \text{on lower surface} \end{cases}. \tag{10.10}$$

These functions are used to close the airfoil at the trailing edge, *i.e* at the point (1,0). The shape functions are shown in figures 10.6 and 10.7. By imposing a closed trailing edge, we can evaluate two of the design variables in terms of the remaining four. When we set $y(1)_{upper} = y(1)_{lower} = 0$ we find from (10.7)that

$$y_1(1)_{lower}\xi_1 + y_2(1)_{lower}\xi_2 + y_3(1)_{lower}\xi_3 + y_4(1)_{lower}\xi_4 + y_5(1)_{lower}\xi_5 + y_6(1)_{lower}\xi_6 = 0, \tag{10.11}$$

$$y_1(1)_{upper}\xi_1 + y_2(1)_{upper}\xi_2 + y_3(1)_{upper}\xi_3 + y_4(1)_{upper}\xi_4 + y_5(1)_{upper}\xi_5 + y_6(1)_{upper}\xi_6 = 0. \tag{10.12}$$

Equations (10.11) and (10.12) can be solved for ξ_5 and ξ_6. This yields

$$\xi_5 = -\left[y_1(1)_{lower}\xi_1 + y_2(1)_{lower}\xi_2 + y_3(1)_{lower}\xi_3 + y_4(1)_{lower}\xi_4\right] \tag{10.13}$$

and

$$\xi_6 = [y_1(1)_{lower}\xi_1 + y_2(1)_{lower}\xi_2 + y_3(1)_{lower}\xi_3 + y_4(1)_{lower}\xi_4]\,. \tag{10.14}$$

Thus the design problem is formulated in terms of 4 design variables.

Mathematically, we write the design problem as

$$\max\ C_L(\xi), \tag{10.15}$$

such that

$$C_D \leq 0.01, \tag{10.16}$$

$$0.075 \leq A \leq 0.150. \tag{10.17}$$

The drag computed is wave drag. The lower limit on the area is imposed so that the airfoil does not reduce to a flat plate and thus maintaining structural integrity. The upper limit is imposed so that we avoid analyzing thick, unrealistic airfoils.

As in Joh's formulation, the airfoils are analyzed for $M = 0.75$ flow at $\alpha = 0$ with the Euler equations for a perfect gas. The pressure distributions are obtained using the Euler code. Lift and drag are computed by numerically integrating the pressure on the airfoil surface.

The airfoil solutions are performed on 201x53 C-grids with 121 points on the airfoil. A typical grid is shown in figure 10.8. Curvature corrected boundary conditions formulated by Dadone and Grossman[81] are used to enforce tangency on the airfoil. With the far-field boundary conditions of Thomas and Salas[82], the computational far field is placed roughly 20 chords away.

CHAPTER
ELEVEN

RESULTS: PART II

In this chapter we present results from several optimization problems using response surface methodology. We begin with a presentation of results from several cases involving the quasi-one-dimensional problem studied in Part 1. Next we present results from the inverse design of a bump in transonic channel flow. Lastly, we demonstrate the usefulness of response surfaces in transonic airfoil design.

11.1 Quasi-one-dimensional duct

The results of response surface optimization for the one-dimensional duct are presented in three sections. In the first section the duct is parameterized with one design variable. This allows for a clear demonstration of the methodology as the design space is easy to visualize. In the next section, results for a three design variable parameterization is presented. A comparison is made between the quadratic polynomial response surface (8.1) and the quadratic tensor product response surface (8.2). Following this section, we make a comparison to the derivative-based optimizations presented in Part 1. All results presented in section 11.1 use the Godunov flow solver on a 64 point grid to analyze flow conditions in the duct. The objective function used for the optimization is defined by (1.24).

One Design Variable Case

The design variable is constrained to the positive design space. For the purpose of probing the design space to construct the response surface we must artificially

restrict the design space to insure that the shape of the duct is reasonable, *i.e.* the flow solver can find a steady state flow solution for the duct. We define the first response surface region with the boundaries

$$1.10 \leq \xi \leq 1.70. \tag{11.1}$$

A quadratic of the form

$$P = c_0 + c_1\xi + c_2 + \xi^2, \tag{11.2}$$

fitted with 5 data points is used for the response surfaces. To satisfy the D-optimality condition, the domain is discretized into 13 points evenly spaced along the ξ-axis. There are 1287 combinations of 5 points from the set of 13 as computed from (8.14). Each combination is checked to satisfy the D-optimality condition. In this case, more than one of the 1287 combinations satisfies D-optimality. Of the D-optimality sets we randomly selected the points listed in table 11.1. D-optimal points for response surfaces in later cycles are found by applying the simple linear transformation of the type

$$\xi_i = \xi_{ll} + \hat{\xi}_i(\xi_{ul} - \xi_{ll}), \tag{11.3}$$

where ξ_i, $i = 1, \ldots, N$ are the D-optimal points, the subscript ll refers to the lower limit of the region, ul refers to the upper limit, and $\hat{\xi}_i$ are the D-optimal points in the region defined for a region scaled between 0 and 1. This transformation saves us from running the GA at every cycle of the optimization.

The response surfaces are constructed using the objective function defined by (1.24) and repeated below for convenience.

$$I(\xi) = \frac{1}{2}\sum_{i=2}^{N-1}(\hat{u} - u)_i^2 \Delta x. \tag{11.4}$$

This is the objective function visualized in one dimension in figure 1.6.

The first response surface is found to be

$$P = 1.725 - 2.467\xi + 0.886\xi^2. \tag{11.5}$$

and is graphed along with the objective function in figure 11.1. The response surface model is able to capture the general location of the minimum, but additional response surface cycles are required to locate the minimum accurately. In this case 8 cycles are required to drive the objective function to machine zero. The convergence is shown in figure 11.2. A history of the design variable is shown in figure 11.3. Also drawn in the figure are the limits of the response surface region.

Three Design Variable Case

The three design variable case is again constrained to the positive design space. The first response surface region is defined within

$$\begin{aligned} 1.05 \leq \xi_1 \leq 1.25, \\ 1.30 \leq \xi_2 \leq 1.50, \\ 1.55 \leq \xi_3 \leq 1.75. \end{aligned} \tag{11.6}$$

This region permits a wide variety of duct shapes to be sampled while constraining the search to reasonable shapes. Most shapes in this region have monotonicly increasing area distributions.

In this design case, we make a comparison between the quadratic (8.1) and the quadratic tensor product (8.2) in three design dimensions. The quadratic surface has 10 coefficients and is fitted with 15 data points. The quadratic tensor product, with 27 coefficients, is fitted with 41 data points. Thus each polynomial is 50% overdetermined.

The design space is sampled at D-optimal points to construct the response surface. To locate D-optimal points, the design domain is discretized into a $9 \times 9 \times 9$ evenly spaced grid. The D-optimal points are found using the genetic algorithm code, GRAMPS. In both the quadratic and quadratic tenor product case, the code ran 3 times with different initial random number seeds. Due to the random nature of the algorithm, the repeatability of the GA result confirms that the set of points found are not anomalous. The histories of the GA convergence are documented in figures 11.4 and 11.5. The D-optimal points for the initial region are listed in tables 11.2 and 11.3 for the quadratic and quadratic tensor product response surfaces respectively.

Several design cycles are required to converge the design. D-optimal points for later cycles are found by applying a linear transformation similar to (11.3) to the D-optimal points listed in tables 11.2 and 11.3.

Optimization with the quadratic polynomial proceeded for 11 cycles before convergence slowed. After the 14th cycle, we switched to derivative-based optimization via the method of conjugate gradients to complete the design. Here we continued to use the objective function defined by (11.4) *i.e.* we did not use any of the techniques discussed in Part 1 to handle the shock wave. The derivative-based optimization was successful because the response surface method was able to get very close to the target and align the design's shock wave with the target's. This is evident in figure 11.6 where the design after the response surface cycles is compare to the target. The complete history of the optimization is shown in figure 11.7.

Optimization with the quadratic tensor product followed a similar pattern. Here, after 4 cycles the convergence slows. After the 10th cycle we switch to derivative-based optimization. Again, the response surface optimization is able to align the shocks so that optimization via the derivative-based methods is successful

without using the techniques of Part 1. The convergence history is shown in figure 11.8.

In a comparison of the performance of the two response surface types, we see that the quadratic polynomial is better able to locate the minimum. Figure 11.7 and 11.8 show that the response surface optimization with the quadratic polynomial is able to converge the minimum to approximately 10^{-7} while with the quadratic tensor product, the objective function only reached 10^{-3}. The quadratic tensor product surface has more terms and thus has more flexibility in modeling the surface, yet the least-squares problem for this surface has a much higher condition number. For the first design cycle, the condition numbers associated with the quadratic polynomial and the quadratic tensor product least-square problem are 4.8×10^7 and 1.0×10^{13}, respectively. For this problem, we conclude that the additional cost of using the quadratic tensor product does not improve convergence. In fact, the high condition number associated with the least-squares fit hinders optimization.

Comparison to Derivative-based Optimization

We now make a comparison to the derivative-based optimizations performed in Part 1. In particular we make the comparison to the optimization with the objective function defined with the coordinate-straining transformation and shock penalty. This method tended to have better convergence than the shock-fitting technique.

In the single design variable formulation, the conjugate gradient optimization converged the design to the target in 3 iterations whereas the response surface method took 8 cycles to reach the same level of convergence. In terms of flow solutions, the response surface method used 5 to construct the surface and one to evaluate the quality of the design at the response surface minimum. Thus a

total of 48 flow solutions were needed in the optimization. On the other hand, the derivative-based optimization, using the adjoint method, requires the equivalent of two flow solutions per iteration to compute the gradient. The line search procedure requires a minimum of three more function evaluations, each needing a flow solution. Thus the derivative-based optimization used a minimum of 15 flow solutions.

For this simple design case, the derivative-based method proved to be the cheaper method. However, the use of parallel computers can improve the attractiveness of the response surface method. Each function evaluation can be sent to a separate processor. As will be demonstrated in the airfoil design problem, the parallel machine can be used to obtain all solutions required to construct the surface at the approximate cost of only one. This would effectively reduce the cost of the design via response surfaces to 16 flow solutions.

The best derivative-based optimization result for the three design variable formulation used finite differences to compute the derivatives. With forward-differences, each iteration would require four solutions to compute gradients, plus at least three more for the line search. The 12 iterations to complete the optimization used at least 72 flow solutions. The response surface, using the quadratic polynomial used 224 flow solutions to achieve almost the same level of optimization. However, with parallel computers, the response surface optimization could have been reduced to 28 equivalent flow solutions. Likewise, the parallel computer could reduce the cost of computing derivatives as a processor can be assigned to compute a derivative. The effective number of solutions for a derivative-based method, taking advantage of coarse grain parallelization is 60.

While response surface methods can compete with derivative based optimizations with the use of parallel machines another key advantage especially in transonic design is that no special handling of the shock is necessary in response surface

methodology. The black-box type approach to optimization with response surface significantly reduces set-up time.

11.2 Bump in a Transonic Channel Flow

The bump inverse design problem has no bounds on the values of the design variables, provided the area constraints (10.6) are satisfied. The first step is to narrow the field to a workable size to commence the response surface algorithm. We note the symmetry of the shape functions 1 and 4, and 2 and 3. Because of this symmetry we choose the limits of ξ_1 and ξ_4 to be the same, as with ξ_2 and ξ_3. We also assume the optimum will be of order 1. Thus we arbitrarily set the first response region to

$$\begin{aligned} 0.25 \leq \xi_1 \leq 0.50, \\ 0.50 \leq \xi_2 \leq 1.00, \\ 0.50 \leq \xi_3 \leq 1.00, \\ 0.25 \leq \xi_4 \leq 0.50. \end{aligned} \tag{11.7}$$

The response surface selected for each cycle is the quadratic curve

$$\begin{aligned} P = & c_0 + c_1\xi_1 + c_2\xi_2 + c_3\xi_3 + c_4\xi_4 + c_5\xi_1^2 + c_6\xi_1\xi_2 \\ & + c_7\xi_1\xi_3 + c_8\xi_1\xi_4 + c_9\xi_2^2 + c_{10}\xi_2\xi_3 + c_{11}\xi_2\xi_4 \\ & + c_{12}\xi_3^2 + c_{13}\xi_3\xi_4 + c_{14}\xi_4^2. \end{aligned} \tag{11.8}$$

We chose 23 D-optimal data points (roughly 1.5 $\times$ the number of coefficients) to construct the response surfaces for each cycle.

D-optimal points for fitting the response surface region with (11.8) is found using GRAMPS. For each cycle, the design space is discretized with 6 points in

each direction to define 1296 candidate points. The list of candidate points is reduced by removing those which do not satisfy the area constraint. This implies that the design space is no longer a hypercube, but rather has an irregular shape. We cannot use a simple linear transformation as we had in the previous design to locate the D-optimal points for future cycles as the shape of the region changes from cycle to cycle. Thus the GA is rerun for each new response region at the start of a cycle.

The GA works with a population of 5 point designs over 5000 generations. After the first cycle, points selected for analysis are not necessarily D-optimal. Any analyzed design points from previous cycles which are also elements of the newest response region are reused in later cycles. Thus the GA is set to maximize $|A^T A|$ with any pre-selected points as elements in the least-squares matrix.

Three response cycles are performed for this inverse design. The details are listed in tables 11.4, 11.5, and 11.6. Each table contains the limits of the response surface region, the D-optimal data points, and the minimum of the response surface. A history of the objective function is shown in figure 11.9. The convergence of the design variables as well as the movement of the upper and lower limit of response surface region is shown in figure 11.10 and 11.11.

The results of the first response surface are encouraging. The minimum of the response surface is also the most attractive design of the 23 analyzed to make the response surface. However, the minimum of the second response surface, when analyzed with the Euler solver is found to produced a design that is surpassed in quality by 20 previously analyzed designs! The third response surface was created about this point despite its high objective function value. The design improved at the conclusion of the third cycle to yield a design with an objective function lower then the previous iteration, but higher than the first.

The pressure distribution of the designs after each cycle is compared to the target in figures 11.12, 11.13, 11.14. The corresponding shapes compared to the target is shown in figures 11.15, 11.16, 11.17. The first response surface cycle matches the pressure distribution very well except at one point at the shock. This corresponds to a slight underprediction of the bump height.

Matching the shock precisely is a heavy requirement of the objective function with differences carrying a heavy penalty. This is an unfortunate consequence of the least squares objective function. The design with the lowest objective function is encountered during the construction of the second response cycle (pt.4 in table 11.5). A view of the pressure distribution and shape is shown in figure 11.18 and 11.19. Again, despite its low objective function, the shape is relatively far from the intended design. The design happened to agree at the shock better than most other designs. It, however, is not very good away from the shock, hence the poor agreement with the intended shape.

The conclusion that we draw from this design is that the least-squares objective function, typical for inverse design, places too much weight on locating the shock. This tends to make bad designs look more impressive than they really are if the shock placement is good. Likewise, very good designs are penalized for slight errors near the shock. Further cycles are not considered for this design as the first response surface result is considered acceptable in light of the shock requirement.

11.3 Transonic Airfoil Design

The transonic airfoil design problem, like the previous problems, enforces no bounds on the variables. Any set of values in $\Re^4$ is feasible provided the drag constraint (10.16) and the area constraints (10.17) are satisfied. To begin the response surface optimization we confine the search to

$$-1.00 \leq \xi_1 \leq 1.00 \qquad i = 1, \ldots, 4 \tag{11.9}$$

Again we use the quadratic function defined by (11.8) to create our response surfaces. The D-optimal set of points for the quadratic are selected by discretizing the domain into 5 levels in each direction. Only points which satisfy the area constraint are considered as candidate points. This changes the shape of the domain from a hypercube to an irregular-shaped domain forcing the GA to be rerun at the start of every design cycle *i.e.* we cannot apply a transformation to a previously found set of D-optimal points.

There is no way to enforce the drag constraint at the point selection level. The drag requires an Euler analysis which can only be afforded at the D-optimal points. Thus the D-optimal points are selected without regard to whether they satisfy the drag constraint or not.

The drag constraint is enforced loosely using a surface fit. That is, the drag is fitted with the data collected from the Euler analyzes at the D-optimal points. In this case, the drag is fitted using (11.8) since the D-optimal points are optimized for a quadratic fit. The response surface modeling the lift is maximized subject to the constraints on the area and the drag modeled by the drag surface. We say the drag constraint is loosely enforced because the optimum point will satisfy the drag response surface constraint, but upon analysis we may find that the true drag is slightly violated.

In this design, we ran 5 response surface cycles. In each cycle we evaluate 23 designs to obtain data for the lift and drag response surfaces. The Euler analyses are performed using ErICA on the Virginia Tech 28 node Intel Paragon XP/E parallel machine. We developed a front end for the code whereby each of 23 processors are assigned to perform one Euler analysis.

The parallel code was analyzed for speed-up and efficiency using a test problem defined by the target flow for the transonic bump problem. The test timed how

long the code took to obtain 24 solutions. Speed-up is defined as

$$Speed\ up = \frac{t_p}{t_1}, \tag{11.10}$$

where t_p is the time to get 24 solutions on p processors and t_1 is the time to get 24 solutions on one processor. Efficiency is defined as

$$Efficiency = \frac{t_p}{t_1 p}. \tag{11.11}$$

The code ran with 1, 4, 6, 8, 12, and 24 nodes with the results shown graphically in figure 11.20 and 11.21. On one node, the code ran the Euler analysis sequentially 24 times; on 4 nodes, the code ran through a loop of length 6, with 4 Euler analyses performed in parallel each time.

In practice, each processor solves a slightly different problem. One problem may take more iterations to reach steady state then others. In such a case, the processors which finishes first remains idle until the last problem is finished.

The data from the five cycles are listed in tables 11.7 through 11.11. Each table contains the limits of the response surface region, the D-optimal design points, and the maximum of the lift surface subject to the drag surface and area constraint. A history of the objective function is plotted in figure 11.22 and the convergence of the design variables shown in figures 11.23 and 11.24.

Although the lift peaked in the fourth cycle, it violated the true drag constraint by 3.1%. Despite the lower lift in cycle 5, the drag constraint is satisfied to a closer degree (1.5%). The design variables do not seem to be converged completely as there are still small variations in the second and third decimal place. Nevertheless, the optimization stopped because of the small variation in the lift. The shape of the final design is shown in figure 11.25. The pressure distribution on the surface is shown in figure 11.26. Pressure contours of the flow field are shown in figure 11.27.

CHAPTER
TWELVE

CONCLUSIONS: PART II

In this section of the dissertation, we presented a method of optimization using response surfaces. The response surface method involves curve fitting the design space with a simple polynomial function. The surface is minimized using conventional techniques. Minimization of the response surface often yields a design in the vicinity of the optimum. Performing other response surface cycles, or continuing the optimization using derivative-based methods generally leads to the optimum design. The first problem involves the design of a duct with mixed supersonic/subsonic flow. The second involves the design of a bump in a channel of transonic flow, and the final example is a design of a transonic airfoil. Each problem has a flow with a shock wave. The interaction of the shock wave with the discretization of the flow field introduces noise in the objective function.

Often in aerodynamic shape design, optimization via derivative-based methods is expensive and prone to failures due to numerical inaccuracies in computing the objective function. These inaccuracies are amplified in the calculation of the derivatives. The advantage of response surface methodology lies in that minimization of the response surfaces can be performed very cheaply and robustly since they are smooth analytic functions.

The choice of response surface can have a profound impact on the success of the optimization. Large polynomials have the flexibility to model many objective functions, but suffer from ill-conditioning in the least-squares problem. While the ill-conditioning may not prevent the polynomial from modeling the function well, it

may cause difficulties in the optimization of the response surface. A simple quadratic polynomial worked well in the design cases presented.

While response surfaces are cheap to optimize, they can be expensive to make. The expense can be reduced by carefully selecting points to use in the fitting procedure. For this we use D-optimal points. D-optimal points can produce a high quality fit with few function evaluations. Locating D-optimal points, however, is not trivial as it leads to an optimization of a large, integer subproblem. However, an effective and efficient genetic algorithm was developed to solve the subproblem.

The cost of generating the response surface can also be reduced by simple coarse-grained parallelization. Evaluating data points for surface construction can be computed simultaneously by assigning each point to a separate processor. The use of parallel computers makes response surfaces an attractive alternative to derivative-based optimization.

The dissertation ended with the presentation of the design results. The first case involved the design of the cross sectional area of a duct to match a given velocity distribution. The results for several cases varying in number of design variables were encouraging as the target shape was recovered using several response surface cycles or a mixture of response surfaces and derivative-based optimization. In a comparison to pure derivative-based optimization, we found that response surface methodology has the advantage that no special treatment of the shock wave was necessary. In the derivative-based optimization, the objective function had to be smoothed, either by shock-fitting or by coordinate-straining. We also found the response surface methodology could be made competitive with derivative-based optimization from a CPU point of view by taking advantage of coarse-grained parallelization.

Optimization of the second problem was performed very quickly using response surfaces. In this inverse design problem, where the pressure over the bump was

matched to a target, we found that the least-square objective function made some designs appear much worse than they really were. This is a result of a slight smearing of the shock wave. As a result the target could not be recovered precisely.

Finally, results for a transonic airfoil are presented. The design required 120 Euler analyses, 24 at each iteration. However, 23 analyses could be performed at once on the Virginia Tech Intel Paragon. This effectively reduced the cost to 10 Euler analyses. For the design of a Mach 0.8 airfoil with a maximum 10 drag counts, the lift coefficient at zero degrees angle of attack was maximized to 0.62.

Weaknesses of the response surface methodology are associated with what is termed the "curse of dimensionality." As the dimension of the design space increases, the feasibility of optimization by response surfaces decreases. In large design dimensions, the number of coefficients required to describe the response surface also becomes large. To achieve a well-conditioned fit to the polynomial requires many function evaluations and hence many CFD solutions. Giunta *et al.*[83] began addressing this issue by taking two approaches. First, they use a variable-complexity approach whereby refined, expensive codes and simple, cheaper codes are used together to develop the response surface. Second, they use regression analysis and analysis of variance to eliminate terms of the polynomial response surface that are not insignificant. In this way a smaller polynomial, and hence a cheaper one to construct, can be used to locate the general area of the minimum. Future work in response surface methodology will include using these techniques in higher dimensional problems.

A strength of response surface optimization is the ease at which it can be applied to a variety of problems. Design problems which may involve three-dimensional flows or viscous flows can easily be optimized using the same codes developed for the one-dimensional duct problem. The only modification which

needs to be done is to swap the one-dimensional flow solver with a flow solver appropriate for the current design problem *i.e.* a three-dimensional or viscous flow solver.

REFERENCES

[1.] Anderson, W.K., *Implicit Multigrid Algorithms for the Flux Split Euler Equations*, Ph.D. Dissertation, Mississippi State University, Aug. 1986.

[2.] Godfrey, A.G., *Topics on Spatially High-order Accurate Methods and Preconditioning for the Navier Stokes Equations with Finite-rate Chemistry*, Ph.D. Dissertation, VPI&SU, Dec. 1992.

[3.] Nieuwland, G.Y., "The Computation by Lighthill's Method of Transonic Potential Flow Around a Family of Quasi-elliptical Aerofoils," NLR TR T.83, 1964.

[4.] Nieuwland, G.Y., "Transonic Potential Flow Around A Family of Quasi-elliptical Aerofoil Sections," NLR TR T.172, 1967.

[5.] Takanaski, S., "A Method of Obtaining Transonic Shock-Free Flow Around Lifting Aerofoils," Transactions, Japan Society for Aeronautical and Space Sciences, vol. 16, No. 34, pp. 246-263, 1973.

[6.] Boerlstoel, J.W., and Huizing, G.H., "Transonic Airfoil Design by an Analytic Hodograph Method," AIAA Paper 74-539, 1974.

[7.] Hobson, D.E. "Shock-free Transonic Flow in Turbomachinery Cascades," University of Cambridge, UK, CUED/A TURBO/TR 65, 1974.

[8.] Sobieczky, H., "Entwurf überkritischer Profile mit Hilfe der Rhoe-Elektrische Analogie," DLR-FB 75-43, 1975.

[9.] Eberle, A "Eine Exakte Hodographenmethode zum Entwurf überkritische Profile," MBB UFE 1168-75 ö, 1975.

[10.] Bauer, F., Garabedian, P., and Korn, D., *Supercritical Wing Section III*, Springer Verlag, 1977.

[11.] Schrier, S., *Compressible Flow,* John Wiley and Sons, New York, 1982

[12.] Sobieczky, H., Fung, K.Y., and Seebass, A.R. "A New Method for Designing Shock-free Transonic Configurations," AIAA Paper No. 78-1114, 1978.

[13.] Yu, N.J., "Efficient Transonic Shock-free Wing Redesign Procedure Using a Fictitious Gas Method," AIAA Paper No. 79-0075, 1979.

[14.] Eberle, A., "Transonic Flow Computations by Finite Elements: Airfoil Optimization and Analysis," in *Recent Developments in Transonic and Experimental Fluid Mechanics*, Springer Verlag, 1979.

[15.] Fung, K.Y., Sobieczky, H., and Seebass, A.R., "Shock-free Wing Design," AIAA Paper No. 79-1557, 1979.

[16.] Rai, P., Miranda, L.R., and Seebass, A.R., "A Cost Effective Method for Shock-free Supercritical Wing Design," AIAA Paper No. 81-0383, 1981.

[17.] Fung, K.Y., Seebass, A.R., Dickson, L.J., and Pearson, C.F., "An Effective Algorithm for Shock-free Wing Design," AIAA Paper No. 81-1236, 1981.

[18.] Holst, T.L., Sloof, J.W., Yoshihara, H., and Ballhaus, W.F.Jr., "Computational Procedures in Transonic Aerodynamic Design," AGARDograph No. 266, pp. 52-67, 1982.

[19.] Dulikravich, G.S., "Aerodynamic Shape Design," AGARD Report No. 780, Nov. 1990.

[20.] Lighthill, M.J., "A New Method of Two-dimensional Aerodynamic Design," ARC R&M 2112, 1945.

[21.] Van Ingen, J.L., "A Program for Airfoil Section Design Utilizing Computer Graphics," AGARD Short Course Notes, 1969.

[22.] Arlinger, B., "An Exact Method of Two-dimensional Airfoil Design," TN67, Saab, Sweden 1970.

[23.] Strand, T., "Exact Method of Designing Airfoils with Given Velocity Distribution in Incompressible Flow," *Journal of Aircraft*, 10 651-659, 1973.

[24.] Woods, L.C., "Airfoil Design in Two-dimensional Subsonic Flow," R & M 2845 Aeronautical Research Council, London England, 1952.

[25.] Tranen, T.L., "A Rapid Computer Aided Transonic Airfoil Design Method," AIAA Paper 74-501, 1974.

[26.] Carlson, L.A., "Transonic Airfoil Analysis & Design Using Cartesian Coordinates," *Journal of Aircraft*, 13, 1976.

[27.] Shankar, V., "A Full Potential Inverse Method Based on a Density Linearization Scheme for Airfoil/Wing Design," AIAA Paper 81-1234, 1981.

[28.] Volpe, G., & Melnik, R.M., "The Design of Transonic Airfoils by a Well-posed Inverse Method," *Journal for Numerical Methods in Engineering*, 22, pp. 341-361.

[29.] Sloof, J.W., "Computational Methods for Subsonic and Transonic Aerodynamic Design," AGARD Report No. 712, pp. 3.2-3.40, 1983.

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

[31.] Mosetti, G., Poloni, C., "Aerodynamic Shape Optimization by Means of a Genetic Algorithm," Proceedings of the 5th Int. Symp. on Computational Fluid Dynamics, Sendai, vol. II, JSCFD, 1993.

[32.] Chen, H.Q., Periaux, J., Stoufflet, B., Mantel, B., "An Implementation of Genetic Algorithms for Aerodynamic Optimum Shape Design, to appear.

[33.] Gage, P.J., Kroo, I.M., Sobieski, I.P., "A Variable-Complexity Genetic Algorithm for Topological Design," AIAA Paper 94-4413, Sept. 1994.

[34.] Elder, J.F., "Global R^d Optimization when Probes are Expensive: the GROPE Algorithm," Proceeding IEEE International Conference on System, Man and Cybernetics, Chicago, Illinois, Oct 18-20, 1992.

[35.] Kushner, H.J., " A New Method of Locating the Maximum of an Arbitrary Multipeak Curve in the Presence of Noise," *Journal of Basic Engineering*, March 97-106, 1964.

[36.] Montgomery, D.C., *Design and Analysis of Experiments*, John Wiley and Sones, New York, 1976.

[37.] Toropov, V.V., "Simulation Approach to Structural Optimization," *Structural Optimization*, vol. 1, pp. 37-46, 1989.

[38.] Toropov, V.V., Filatov, A.A., Polynkin, A.A., "Multiparameter Structural Optimization Using FEM and Multipoint Explicit Approximations," *Structural Optimization*, vol. 6, pp. 7-14, 1993.

[39.] Toropov, V.V., "Multipoint Approximation Method for Structural Optimization and Identification," Proceedings of The World Congress on Optimal Design of Structural Systems, vol. 1, Rio de Janeiro, Aug. 1993.

[40.] Toropov, V.V., Van der Giessen, E., "Parameter Identification for Nonlinear Constitutive Models: Finite Element Simulation - Optimization - Nontrivial Experiments," Proceedings of the IUTAM Symposium on Optimal Design with Advanced Materials, Lyngby, Denmark, Aug. 1992.

[41.] Toropov, V.V., Markin, V.L., Carlsen, Henrik, "Discrete Structural Optimization Based on Multipoint Explicit Approximations," Proceedings of the IUTAM Symposium on Discrete Structural Optimization, Zakopane, Poland, Aug. 1993.

[42.] Polynkin, A.A., Van Keulen, F., and Toropov, V.V., "Optimization of Geometrically Nonlinear Shells," extended abstracts of the Third World Congress on Computational Mechnaics, Chiba, Japan, Aug. 1994.

[43.] Van Keulen, F., Toropov, V.V., and Polynkin, A.A., "Optimization of Geometrically Nonlinear Shell Structures Using Multi-meshing and Adaptivity," AIAA-94-4361 Sept. 1994.

[44.] Markine, V.L., Meijers, P., Meijaard, J.P., and Toropov, V.V., "Multilevel Optimization of Dynamic Behaviour of a Linear Mechanical System with Multipoint Approximation," LTM 1054, Nov. 1994.

[45.] Giunta, A.A., Dudley, J.M., Narducci, R.P., Grossman, B., Haftka, R.T., Mason, W.H., Watson, L.T., "Noisy Aerodynamic Response and Smooth Approximation in HSCT Design," AIAA Paper 94-4376, Sept. 1994.

[46.] Healy, M.J., Kowalik, J.S., Ransay, J.W., "Airplane Engine Selection by Optimization on Surface Fit Approximations," *Journal of Aircraft*, vol. 12, No. 7, 1975, pp. 593, 599.

[47.] Gill, Murray & Wright, *Practical Optimization,* Academic Press, New York, 1981.

[48.] Narducci, R., Grossman, B., Haftka, R.T., "Sensitivity Algorithms for an Inverse Design Problem Involving a Shock Wave," AIAA Paper 94-0096, Jan. 1994, also to appear in *Inverse Problems in Engineering.*

[49.] Frank, P.D., Shubin, G.R., "A Comparison of Optimization-based Approaches for a Model Computational Aerodynamics Design Problem, " *Journal of Computational Physics*, vol. 98, 1992, pp. 74-89.

[50.] Frank, P.D., Shubin, G.R., "A Comparison of the Implicit Gradient Approach and the Variational Approach to Aerodynaimc Design Optimization," Applied

Mathematics and Statistics Technical Rept. AMS-TR-163 Boeing Computer Services, Seattle, WA, April 1991.

[51.] Iollo, A., Salas, M., Ta'asan, S., "Shape Optimization Governed by the Euler Equations using an Adjoint Method," ICASE Report No. 93-78.

[52.] Shenoy, A., Cliff, E., "An Optimal Control Formulation for a Flow Matching Problem," AIAA Paper 94-4306.

[53.] Wu, X., Cliff, E., and Gunzburgwer, M.D., "An Optimal Design Problem for a Two-dimensional Flow in a Duct," ICAM 94-4376.

[54.] Borggaard, J., Burns, J.A., Cliff, E., and Gunzburger M., "Sensitivity Calculations for a 2-D, Inviscid, Supersonic Forebody Problem," ICASE Report No. 93-13, March 1993.

[55.] Borggaard, J., Burns, J., "A Sensitivity Equation Approach to Optimal Design of Nozzles," AIAA Paper 94-4274.

[56.] Jameson, A., "Aerodynamic Design via Control Theory," ICASE Report No. 88-64, MAE Report No. 1824, *Journal of Scientific Computing*, vol. 3, 1988 pp. 233-260.

[57.] Jameson, A., "Computational Algorithms for Aerodynamic Analysis and Design," MAE Report 1966, Dec. 1992.

[58.] Reuther, J., Jameson, A., "Control Theory, Based Airfoil Design for Potential Flow and a Finite-Discretization," AIAA Paper 94-0499, Jan. 1994.

[59.] Jameson, A., "Computational Algorithms for Aerodynamic Analysis and Design," MAE Report 1966, Dec. 1992.

[60.] Reuther, J., Jameson, A., "Control Theory Based Airfoil Design using the Euler Equations," AIAA Paper 94-4272.

[61.] Joh, C -Y., Grossman, B., Haftka, R.T., "Design Optimization of Transonic Airfoil," *Engineering Optimization*, 21, No. 1, 1993, pp. 1-20.

[62.] Gilmore, P., Kelley, C.T., "An Implicit Filtering Algorithm for Optimization of Functions with Many Local Minima," *SIAM Journal on Optimization*, Vol. 5, No. 2, May 1995, pp. 269-285.

[63.] Stoneking, D., Bilbro, G., Trew, R., Gilmore, P., and Kelley, C.T., "Yield Optimization Using a GaAs Process Simulator Coupled to a Physical Device

Model," in Proceedings IEEE, Cornell Conference on Advanced Concepts in High Speed Devices and Circuits, IEEE, 1991, pp. 374-383.

[64.] Winslow, T.A., Trew, R.J., Gilmore, P., and Kelley, C.T., "Doping Profiles for Optimum Class B Performance of GaAs Mesfet Amplifiers," in Proceedings IEEE/ Cornell Conference on Advanced Concepts in High Speed Devices and Circuits, IEEE, 1991, pp. 188-197.

[65.] Nixon, D., "Perturbations of a Discontinuous Transonic Flow," *AIAA Journal*, 16, No. 1, 1978, pp. 47-52.

[66.] Hirsch, C., *Numerical Computation of Internal and External Flows*, vol. 1, John Wiley & Sons, New York, 1989.

[67.] Strahara, S.S., "A Rapid Approximation Procedure for Nonlinear Solutions: Application to Acrodynamic Flows and Design/Optimization Problems," Chapter 18 in *Transonic Aerodynamics*, 81, Progress in Astronautics and Aeronautics, D. Nixon, Ed., AIAA, New York, 1982.

[68.] Hestenes, M.R., Stiefel, E., "Methods of Conjugate Gradients for Solving Linear Systems," *Journal of Research of the Natural Bureau of Standards*, 49, pp. 409-436, 1952.

[69.] Brent, R.P., *Algorithms for Minimization without Derivatives* Prentice-Hall, New Jersey, 1974.

[70.] Press, W.H., Teukolsky, S.A., Vetterling, W.T., Flannery, B.P., *Numerical Recipes in FORTRAN, The art of Scientific Computing*, Cambridge University Press, New York, 1992.

[71.] Schittkowski, k., "NLPQL, A FORTRAN Subroutine Solving Constrained Nonlinear Programming Problems," Annals of Operations Research, vol. 5, pp. 485-500., 1985-1986.

[72.] Carpenter, W.C. "Effect of Design Selection on Response Surface Performance", NASA CR 4250, 1993.

[73.] Box, M. J. and Draper, N. R., "Factorial Designs, the $|\mathbf{X}^T\mathbf{X}|$ Criterion, and Some Related Matters," Technometrics, vol. 13, No. 4, 1971, pp. 731–742.

[74.] Welch, W.J. "Branch and Bound Search for Experimental Designs Based on D-optimality and Other Criteria," Technometrics 24 1982, 41-48.

[75.] Fedorov, V.V., *Theory of Optimal Experiments*, Academic Press, New York 1972.

[76.] Mitchell, T.J. "An Algorithm for the Construction of 'D-optimal' Experimental Designs," Technometrics 16, No. 2, 1974, pp. 203-210.

[77.] Furuya, H., and Haftka, R.T., "Locating Actuators for Vibration Suppression on Space Trusses by Genetic Algorithms," ASME Winter Annual Meeting, 1993.

[78.] Strang, G. *Linear Algebra and its Applications*, Academic Press, New York, 1976, pp. 96-125.

[79.] Vanderplaats, G.N. "Approximation Concepts for Numerical Airfoil Optimization," *Engineering Optimization*, 21, No. 1, 1983, pp. 1-20.

[80.] Abbott, I.H., & von Doenhoff, A.E., *Theory of wing sections, including a summary of airfoil data* Dover Publications, New York, 1959.

[81.] Dadone, A., & Grossman, B., "Surface Boundary Conditions for the Numerical Solution of the Euler Equations," *AIAA Journal* 32, No. 2, 1994, pp. 285-293.

[82.] Thomas, J.L., & Salas, M.D., "Far-Field Boundary Conditions for Transonic Lifting Solutions to the Euler Equations," AIAA Paper 85-0020, Jan. 1985.

[83.] Giunta, A.A., Narducci, R., Burgee, S., Grossman, B., Mason, W.H., Watson, L.T., & Haftka, R.T., "Variable-Complexity Response Surface Aerodynamic Design of an HSCT Wing," AIAA Paper 95-1886, June, 1995.

	i	x_i	Initial ξ_i	Target ξ_i
Case 1	1	0.50	1.2500	1.3975
Case 2	2	0.25	1.0848	1.1586
	3	0.50	1.2500	1.3975
	4	0.75	1.5627	1.6364

Table 5.1: Initial and target design conditions.

	Forward Difference	Central Difference	Direct	Adjoint
$\partial I/\partial \xi$	-0.2849	-0.2849	-0.2998	-0.2799
I_{final}	8.237×10^{-4}	8.237×10^{-4}	8.237×10^{-4}	8.237×10^{-4}
Iterations	3	3	3	3

Table 5.2: Comparison of design sensitivities and convergence of the continuous approach using the exact flow solver. One design variable case using the initial design point described in table 5.1; $I_o = 5.436 \times 10^{-2}$.

	Forward Difference	Central Difference	Direct	Adjoint
$\partial I/\partial \xi_1$	0.0177	0.0177	0.0199	0.0167
$\partial I/\partial \xi_2$	-0.1951	-0.1951	-0.2067	-0.1916
$\partial I/\partial \xi_3$	-0.1972	-0.1972	-0.2062	-0.1931
I_{final}	$8.236\text{x}10^{-4}$	$8.236\text{x}10^{-4}$	$8.256\text{x}10^{-4}$	$8.240\text{x}10^{-4}$
Iterations	31	35	28	16

Table 5.3: Comparison of design sensitivities and convergence of the continuous approach using the exact flow solver. Three design variable case using the initial design point described in table 5.1; $I_o = 5.436\text{x}10^{-2}$.

	Forward Difference	Central Difference	Direct	Adjoint
$\partial I/\partial \xi$	-0.4286	-0.4288	-0.2915	-0.2890
I_{final}	8.129×10^{-4}	8.129×10^{-4}	8.129×10^{-4}	8.129×10^{-4}
Iterations	3	3	4	4

Table 5.4: Comparison of design sensitivities and convergence of the continuous approach using the Godunov flow solver. One design variable case using the initial design point described in table 5.1; $I_o = 5.3120 \times 10^{-2}$.

	Forward Difference	Central Difference	Direct	Adjoint
$\partial I/\partial \xi_1$	0.0440	0.0440	0.0232	0.0196
$\partial I/\partial \xi_2$	-0.3137	-0.3137	-0.2060	-0.2019
$\partial I/\partial \xi_3$	-0.2776	-0.2776	-0.1944	-0.1937
I_{final}	$8.389\text{x}10^{-4}$	$8.342\text{x}10^{-4}$	$9.417\text{x}10^{-4}$	$8.827\text{x}10^{-4}$
Iterations	19	36	13	20

Table 5.5: Comparison of design sensitivities and convergence of the continuous approach using the Godunov flow solver. Three design variable case using the initial design point described in table 5.1; $I_o = 5.3120\text{x}10^{-2}$.

	Target	Forward Difference	Central Difference	Direct	Adjoint
ξ_1	1.3975	1.3994	1.3994	1.3994	1.3994
ξ_1	1.1586	1.1586	1.1587	1.1606	1.1572
ξ_2	1.3975	1.3993	1.3993	1.3997	1.3993
ξ_3	1.6364	1.6354	1.6354	1.6449	1.6322

Table 5.6: Final design parameters for the continuous approach using exact flow solutions.

	Forward Difference	Direct
Exact Solver	-0.67318	-0.70355
Godunov Solver	-0.95425	-0.68037

Table 5.7: Continuous approach shock sensitivities computed with the exact and Godunov flow solvers.

	Target	Forward Difference	Central Difference	Direct	Adjoint
ξ_1	1.3975	1.3969	1.3969	1.3969	1.3969
ξ_1	1.1586	1.1601	1.1600	1.1499	1.1723
ξ_2	1.3975	1.3982	1.3981	1.4010	1.3983
ξ_3	1.6364	1.6727	1.6693	1.7089	1.6890

Table 5.8: Final design parameters for the continuous approach using Godunov flow solutions.

	Forward Difference	Central Difference	Direct	Adjoint
$\partial I/\partial \xi$	-0.5495	-0.5496	-0.5495	-0.5495
I_{final}	1.045×10^{-10}	1.153×10^{-11}	1.048×10^{-10}	1.048×10^{-10}
Iterations	3	3	3	3

Table 5.9: Comparison of design sensitivities and convergence of the discrete approach using the Godunov flow solver. One design variable case using the initial design point described in table 5.1; $I_o = 3.936 \times 10^{-2}$.

	Forward Difference	Central Difference	Direct	Adjoint
$\partial I/\partial \xi_1$	0.0834	0.0834	0.0834	0.0834
$\partial I/\partial \xi_2$	-0.4248	-0.4248	-0.4248	-0.4248
$\partial I/\partial \xi_3$	-0.3367	-0.3367	-0.3367	-0.3367
I_{final}	$8.675\text{x}10^{-9}$	$5.303\text{x}10^{-10}$	$5.186\text{x}10^{-11}$	$5.186\text{x}10^{-11}$
Iterations	12	12	55	55

Table 5.10: Comparison of design sensitivities and convergence of the continuous approach using the Godunov flow solver. Three design variable case using the initial design point described in table 5.1; $I_0 = 3.9363\text{x}10^{-2}$.

	Forward Difference	Central Difference	Direct	Adjoint
$\partial I/\partial \xi$	-0.4224	-0.4224	-0.4224	-0.4224
I_{final}	7.865×10^{-13}	7.715×10^{-13}	7.717×10^{-13}	7.717×10^{-13}
Iterations	3	3	3	3

Table 5.11: Comparison of design sensitivities and convergence of the discrete approach using the artificial flow solver. One design variable case using the initial design point described in table 5.1; $I_o = 3.9363 \times 10^{-2}$.

	Forward Difference	Central Difference	Direct	Adjoint
$\partial I/\partial \xi_1$	0.0554	0.0554	0.0554	0.0554
$\partial I/\partial \xi_2$	-0.3203	-0.3203	-0.3203	-0.3203
$\partial I/\partial \xi_3$	-0.2599	-0.2599	-0.2599	-0.2599
I_{final}	5.147×10^{-9}	5.591×10^{-9}	5.595×10^{-9}	5.595×10^{-9}
Iterations	11	11	11	11

Table 5.12: Comparison of design sensitivities and convergence of the discrete approach using the artificial viscosity flow solver. Three design variable case using the initial design point described in table 5.1; $I_o = 4.0280 \times 10^{-2}$.

	Target	Forward Difference	Central Difference	Direct	Adjoint
ξ_1	1.3975	1.3975	1.3975	1.3975	1.3975
ξ_1	1.1586	1.1586	1.1586	1.1586	1.1586
ξ_2	1.3975	1.3975	1.3975	1.3975	1.3975
ξ_3	1.6364	1.6370	1.6363	1.6364	1.6364

Table 5.13: Final design parameters for the discrete approach using Godunov flow solutions.

	Target	Forward Difference	Central Difference	Direct	Adjoint
ξ_1	1.3975	1.3975	1.3975	1.3975	1.3975
ξ_1	1.1586	1.1585	1.1585	1.1584	1.1584
ξ_2	1.3975	1.3975	1.3975	1.3975	1.3975
ξ_3	1.6364	1.6367	1.6368	1.6368	1.6368

Table 5.14: Final design parameters for the discrete approach using artificial viscosity flow solutions.

	$\lvert \mathbf{A}^T\mathbf{A} \rvert$	V
D-optimal	5.960×10^{-8}	0.10705
D-optimal/center hybrid	9.3132×10^{-10}	0.11406
Center	1.4552×10^{-11}	0.14466
Diamond	3.5527×10^{-15}	0.49095
Interior	8.6736×10^{-19}	0.13738

Table 8.1: Comparison of different sets of points for response surface construction.

Control Point	Shape 1	Shape 2	Shape 3	Shape 4
1	(0,0)	(0,0)	(0,0)	(0,0)
2	(0,0)	(0,0)	(0,0)	(0,0)
3	(0,0)	(0,0)	(0,0)	(0,0)
4	(0.2,0.1)	(0.4,0.1)	(0.6,0.1)	(0.8,0.1)
5	(1,0)	(1,0)	(1,0)	(1,0)
6	(1,0)	(1,0)	(1,0)	(1,0)
7	(1,0)	(1,0)	(1,0)	(1,0)

Table 10.1: B-spline control points for the bump shape functions

Index	ξ
1	1.1
2	1.35
3	1.4
4	1.65
5	1.7

Table 11.1: D-optimal points for a quadratic polynomial response surface in the region $1.10 \leq \xi \leq 1.70$.

Index	ξ_1	ξ_2	ξ_3
1	1.15	1.50	1.55
2	1.05	1.40	1.55
3	1.25	1.30	1.55
4	1.05	1.50	1.55
5	1.15	1.30	1.55
6	1.05	1.50	1.65
7	1.05	1.30	1.55
8	1.25	1.30	1.75
9	1.25	1.50	1.75
10	1.25	1.50	1.55
11	1.15	1.40	1.75
12	1.05	1.50	1.75
13	1.05	1.30	1.75
14	1.25	1.40	1.65
15	1.100	1.30	1.65

Table 11.2: D-optimal points for a quadratic polynomial response surface in the region defined by (11.6).

Index	ξ_1	ξ_2	ξ_3
1	1.10	1.30	1.75
2	1.15	1.35	1.55
3	1.05	1.30	1.75
4	1.05	1.40	1.65
5	1.05	1.30	1.70
6	1.20	1.45	1.75
7	1.10	1.35	1.55
8	1.05	1.40	1.60
9	1.25	1.50	1.75
10	1.05	1.30	1.55
11	1.20	1.50	1.55
12	1.25	1.30	1.55
13	1.15	1.45	1.70
14	1.15	1.30	1.70
15	1.25	1.40	1.70
16	1.15	1.30	1.75
17	1.15	1.40	1.65
18	1.25	1.40	1.55
19	1.05	1.30	1.65
20	1.25	1.50	1.55
21	1.05	1.40	1.55
22	1.05	1.50	1.65
23	1.25	1.30	1.75
24	1.25	1.40	1.65
25	1.05	1.35	1.55
26	1.25	1.40	1.75
27	1.20	1.35	1.65
28	1.25	1.30	1.65
29	1.15	1.30	1.65
30	1.20	1.40	1.65
31	1.05	1.50	1.75
32	1.15	1.50	1.75
33	1.10	1.50	1.60
34	1.15	1.30	1.55
35	1.15	1.35	1.75
36	1.15	1.50	1.60
37	1.25	1.50	1.65
38	1.05	1.50	1.55
39	1.15	1.40	1.55
40	1.05	1.40	1.75
41	1.05	1.45	1.65

Table 11.3: D-optimal points for a quadratic tensor product response surface in the region defined by (11.6).

CYCLE 1

	ξ1	ξ2	ξ3	ξ4	OBJ. FUN.	Fit
min	0.2500	0.5000	0.5000	0.2500		
max	0.5000	1.0000	1.0000	0.5000		
1	0.2500	0.7000	0.7000	0.2500	0.120750	0.120750
2	0.2500	0.5000	0.9000	0.2500	0.134100	0.135290
3	0.2500	0.5000	0.5000	0.2500	0.059611	0.059405
4	0.2500	0.9000	0.5000	0.2500	0.167530	0.164010
5	0.2500	0.7000	0.5000	0.2500	0.071307	0.065282
6	0.4000	0.5000	0.5000	0.5000	0.430490	0.430420
7	0.3000	0.5000	0.7000	0.4000	0.343890	0.319350
8	0.3000	0.7000	0.5000	0.4000	0.234070	0.240710
9	0.4000	0.5000	0.5000	0.3000	0.083435	0.091438
10	0.3500	0.5000	0.5000	0.2500	0.052258	0.049939
11	0.2500	0.7000	0.5000	0.4500	0.361840	0.356860
12	0.5000	0.5000	0.5000	0.4000	0.204260	0.206310
13	0.3500	0.8000	0.5000	0.2500	0.163520	0.164870
14	0.4500	0.7000	0.5000	0.2500	0.177760	0.175090
15	0.2500	0.5000	0.7000	0.4500	0.485100	0.487590
16	0.3500	0.5000	0.8000	0.2500	0.090564	0.095880
17	0.2500	0.5000	0.7000	0.2500	0.095948	0.092765
18	0.2500	0.5000	0.5000	0.5000	0.591710	0.568910
19	0.2500	0.8000	0.5000	0.2500	0.095008	0.103040
20	0.2500	0.5000	0.5000	0.4000	0.291560	0.303740
21	0.4500	0.5000	0.7000	0.2500	0.095325	0.097472
22	0.5000	0.5000	0.5000	0.2500	0.103450	0.098904
23	0.2500	0.5000	0.6000	0.5000	0.576560	0.602030
opt	0.3093	0.5266	0.5000	0.2500	0.050654	

Table 11.4: Transonic bump response surface cycle 1 results.

CYCLE 2

	ξ_1	ξ_2	ξ_3	ξ_4	OBJ. FUN.	Fit
min	0.2781	0.4640	0.2500	0.1250		
max	0.3406	0.5890	0.7500	0.3750		
1	0.3094	0.5266	0.5000	0.2500	0.050654	0.078846
2	0.2781	0.4890	0.2500	0.3250	0.482580	0.453080
3	0.3406	0.5890	0.5500	0.1250	0.206320	0.163070
4	0.3406	0.4640	0.7500	0.1750	0.042122	0.041671
5	0.2781	0.5890	0.2500	0.3750	0.109000	0.127200
6	0.3406	0.5890	0.2500	0.2750	0.164520	0.288620
7	0.3406	0.4640	0.2500	0.3750	0.125730	0.115030
8	0.3156	0.4640	0.7500	0.3750	0.307320	0.358560
9	0.2781	0.4890	0.2500	0.1250	1.953000	1.906100
10	0.2781	0.4640	0.5500	0.1250	0.216380	0.365090
11	0.2781	0.4890	0.7500	0.3750	0.329990	0.398010
12	0.3031	0.5640	0.7500	0.1250	0.071167	0.072763
13	0.2781	0.4640	0.7500	0.3250	0.238250	0.059088
14	0.3156	0.5890	0.3500	0.3750	0.101560	-0.032506
15	0.3406	0.5640	0.5500	0.3750	0.194090	0.133920
16	0.2781	0.5890	0.6500	0.2250	0.060254	-0.020028
17	0.3406	0.5890	0.7500	0.1750	0.096851	0.137000
18	0.2781	0.5890	0.7500	0.2250	0.086970	0.126420
19	0.2781	0.5640	0.6500	0.3750	0.274640	0.377010
20	0.2781	0.5140	0.7500	0.1250	0.043917	-0.009066
21	0.3156	0.4640	0.2500	0.3750	0.133730	0.185580
22	0.3406	0.4640	0.2500	0.1250	1.974900	1.929000
23	0.2781	0.5890	0.2500	0.1250	1.348700	1.358100
opt	0.2781	0.4640	0.5454	0.3378	0.129486	

Table 11.5: Transonic bump response surface cycle 2 results

CYCLE 3

	ξ_1	ξ_2	ξ_3	ξ_4	OBJ. FUN.	Fit
min	0.2469	0.4016	0.4829	0.3066		
max	0.3094	0.5266	0.6079	0.3691		
1	0.3094	0.5266	0.5000	0.2500	0.050654	0.052379
2	0.2781	0.4640	0.5454	0.3378	0.129490	0.133200
3	0.2469	0.4266	0.4829	0.3191	0.166440	0.169660
4	0.2469	0.4016	0.5079	0.3066	0.166310	0.166720
5	0.2469	0.4016	0.6079	0.3691	0.325560	0.327660
6	0.2969	0.5266	0.4829	0.3316	0.092381	0.090191
7	0.2469	0.5266	0.6079	0.3691	0.288860	0.283290
8	0.2969	0.5266	0.4829	0.3691	0.131810	0.136380
9	0.2469	0.5266	0.4829	0.3691	0.170600	0.169180
10	0.3094	0.4266	0.4829	0.3066	0.107650	0.106820
11	0.2969	0.5266	0.5579	0.3316	0.099508	0.102540
12	0.2719	0.5266	0.4829	0.3066	0.094712	0.089235
13	0.2469	0.5266	0.5579	0.3316	0.123410	0.128960
14	0.3094	0.4016	0.4829	0.3691	0.176660	0.177730
15	0.3094	0.4766	0.5579	0.3691	0.196820	0.188650
16	0.2594	0.4016	0.5829	0.3066	0.143980	0.138410
17	0.2844	0.5266	0.6079	0.3066	0.092827	0.088056
18	0.2469	0.4016	0.5079	0.3691	0.246970	0.244430
19	0.2844	0.4266	0.6079	0.3691	0.282580	0.283810
20	0.2844	0.5266	0.6079	0.3691	0.260470	0.265350
21	0.2719	0.4516	0.4829	0.3691	0.180080	0.180440
22	0.2469	0.5016	0.6079	0.3066	0.107800	0.110100
23	0.3094	0.4016	0.6079	0.3066	0.105820	0.108190
opt	0.3094	0.5266	0.5370	0.3066	0.082873	

Table 11.6: Transonic bump response surface cycle 3 results

CYCLE 1

	ξ1	ξ2	ξ3	ξ4	CL	Fit	CD	Fit
min	-1.0000	-1.0000	-1.0000	-1.0000				
max	2.0000	2.0000	2.0000	2.0000				
1	1.5000	-1.0000	-1.0000	1.5000	-0.127510	-0.118120	0.003330	0.004050
2	2.0000	-0.5000	-0.5000	0.0000	0.219230	0.248150	0.007730	0.007474
3	-1.0000	1.0000	-1.0000	2.0000	0.198790	0.202190	0.014390	0.014569
4	2.0000	-1.0000	-1.0000	1.0000	-0.049600	-0.076822	0.008170	0.006517
5	0.5000	-0.5000	2.0000	-1.0000	0.741900	0.740290	0.053950	0.052781
6	-0.5000	-1.0000	2.0000	0.0000	0.394050	0.395390	0.012430	0.012121
7	-0.5000	1.5000	-1.0000	1.0000	0.411560	0.413650	0.010060	0.010693
8	-1.0000	-1.0000	2.0000	0.5000	0.312550	0.320040	0.017180	0.018921
9	-1.0000	1.0000	0.5000	0.5000	0.567800	0.563250	0.042680	0.041565
10	2.0000	1.0000	-1.0000	-0.5000	0.703420	0.718100	0.049730	0.050026
11	1.5000	1.0000	-0.5000	-0.5000	0.770360	0.758610	0.059310	0.059235
12	2.0000	-1.0000	1.0000	-1.0000	0.595280	0.584970	0.018190	0.018926
13	0.0000	0.0000	0.0000	1.0000	0.320350	0.309550	0.011600	0.012697
14	1.5000	-1.0000	0.0000	0.5000	0.186830	0.208570	0.004010	0.004090
15	-1.0000	-0.5000	0.0000	2.0000	-0.113120	-0.118980	0.002790	0.002347
16	0.5000	2.0000	-1.0000	0.0000	0.783410	0.768030	0.061300	0.059195
17	0.5000	0.0000	0.0000	0.5000	0.398470	0.385750	0.009580	0.009055
18	0.0000	0.5000	-1.0000	1.5000	0.165600	0.183930	0.002810	0.003827
19	1.5000	2.0000	-1.0000	-1.0000	0.957200	0.947600	0.063120	0.062596
20	0.5000	0.5000	1.0000	-1.0000	0.813050	0.806680	0.037550	0.036771
21	2.0000	-0.5000	-1.0000	0.5000	0.082700	0.072422	0.006210	0.007246
22	-1.0000	-1.0000	0.5000	2.0000	-0.102950	-0.105920	0.005350	0.004361
23	0.5000	2.0000	0.0000	1.0000	0.959710	0.981760	0.091220	0.093628
opt	1.1479	-1.0000	1.5175	-0.9017	0.567800	0.557600	0.006600	0.010000

Table 11.7: Transonic airfoil design response surface cycle 1 results

CYCLE 2

	ξ1	ξ2	ξ3	ξ4	CL	Fit	CD	Fit
min	0.7729	-2.5000	1.1425	-1.2767				
max	1.5229	0.5000	1.8925	-0.5267				
1	0.7729	0.0000	1.1425	-0.7767	0.746970	0.744250	0.050130	0.049905
2	1.3979	0.0000	1.1425	-1.2767	0.842670	0.842250	0.063960	0.063574
3	0.7729	0.0000	1.6425	-1.2767	0.831260	0.829550	0.064450	0.065771
4	1.1479	-0.5000	1.5175	-1.2767	0.749260	0.747040	0.023180	0.023883
5	1.5229	-2.0000	1.7675	-0.5267	0.397390	0.391050	0.003030	0.004726
6	1.2729	0.0000	1.2675	-1.2767	0.838170	0.836990	0.067200	0.068136
7	1.2729	-1.0000	1.8925	-1.2767	0.725750	0.723350	0.027250	0.027581
8	1.3979	-1.0000	1.3925	-0.7767	0.619750	0.622120	0.026590	0.026270
9	0.7729	-0.5000	1.8925	-1.2767	0.777580	0.778500	0.035730	0.035031
10	1.5229	-2.0000	1.8925	-0.7767	0.376590	0.377750	0.000360	0.001487
11	0.8979	-1.5000	1.8925	-0.5267	0.497060	0.498060	0.010670	0.010723
12	1.1479	-1.0000	1.5175	-0.7767	0.605380	0.605260	0.018070	0.017637
13	1.3979	-1.0000	1.8925	-1.2767	0.739290	0.743250	0.041030	0.040440
14	1.5229	-2.0000	1.6425	-0.5267	0.283870	0.287130	0.000090	-0.002234
15	1.1479	-1.0000	1.8925	-1.0267	0.692980	0.692050	0.041130	0.040219
16	1.5229	-1.0000	1.1425	-0.7767	0.541440	0.542560	0.010310	0.011095
17	1.2729	-1.0000	1.1425	-0.5267	0.514750	0.515470	0.010370	0.010582
18	1.5229	-1.0000	1.1425	-0.6517	0.576390	0.574970	0.021080	0.021042
19	0.7729	-1.0000	1.5175	-0.5267	0.522520	0.522210	0.008390	0.008749
20	1.5229	-0.5000	1.2675	-1.2767	0.759960	0.759650	0.030350	0.029801
21	0.7729	0.5000	1.1425	-1.2767	0.885310	0.888910	0.057770	0.057005
22	1.1479	-1.0000	1.3925	-0.5267	0.574250	0.578200	0.026990	0.026604
23	1.1479	-1.0000	1.1425	-0.5267	0.449960	0.447970	0.002200	0.002602
opt	1.0774	-0.9130	1.5272	-0.9204	0.590180	0.588750	0.007970	0.010000

Table 11.8: Transonic airfoil design response surface cycle 2 results

CYCLE 3

	ξ1	ξ2	ξ3	ξ4	CL	Fit	CD	Fit
min	0.9836	-1.2880	1.4335	-1.0141				
max	1.1711	-0.5380	1.6210	-0.8266				
1	1.1711	-0.5380	1.4335	-0.9516	0.723430	0.723710	0.045480	0.045525
2	1.1711	-0.9130	1.5897	-0.8266	0.655270	0.655800	0.034500	0.034477
3	1.0149	-0.5380	1.5585	-1.0141	0.735180	0.738360	0.040100	0.040170
4	0.9836	-1.0380	1.6210	-0.8266	0.560260	0.559440	0.006630	0.006319
5	1.1399	-1.1630	1.6210	-0.8266	0.555070	0.555790	0.007710	0.008092
6	1.0149	-0.9130	1.4960	-0.8266	0.574090	0.573520	0.007750	0.007772
7	1.1399	-0.5380	1.4960	-0.9829	0.730410	0.730270	0.047820	0.047870
8	1.0774	-0.7880	1.6210	-0.8891	0.687810	0.685610	0.038190	0.037770
9	1.1711	-0.9130	1.4960	-0.9204	0.615050	0.616180	0.012960	0.013285
10	1.1711	-1.0380	1.6210	-0.9829	0.594230	0.594310	0.008510	0.008456
11	0.9836	-0.7880	1.6210	-0.8266	0.671280	0.672420	0.034640	0.034825
12	1.1399	-0.7880	1.4335	-0.8266	0.654750	0.654770	0.025840	0.025999
13	0.9836	-0.5380	1.4647	-0.8266	0.700350	0.700070	0.044060	0.044011
14	0.9836	-0.7880	1.4335	-0.8266	0.599220	0.599570	0.010420	0.010592
15	1.0774	-0.6630	1.4335	-1.0141	0.655720	0.656140	0.013820	0.013861
16	0.9836	-0.5380	1.4335	-0.9204	0.707020	0.706820	0.029930	0.029660
17	0.9836	-0.5380	1.4960	-1.0141	0.720990	0.718590	0.027950	0.027820
18	0.9836	-0.7880	1.4960	-0.8891	0.615940	0.617320	0.011620	0.011832
19	0.9836	-0.7880	1.6210	-1.0141	0.651060	0.651240	0.014250	0.014290
20	1.1711	-0.7880	1.5897	-1.0141	0.703620	0.700210	0.030510	0.030170
21	1.1711	-0.7880	1.6210	-1.0141	0.705010	0.707330	0.034140	0.034336
22	0.9836	-0.5380	1.6210	-0.9829	0.733140	0.732410	0.048790	0.048956
23	1.1711	-1.0380	1.4647	-0.8266	0.547420	0.546450	0.005980	0.005514
opt	1.1711	-0.8015	1.4335	-0.9913	0.620440	0.620570	0.010470	0.010000

Table 11.9: Transonic airfoil design response surface cycle 3 results

CYCLE 4

	ξ1	ξ2	ξ3	ξ4	CL	Fit	CD	Fit
min	1.0774	-0.8953	1.3397	-1.0147				
max	1.2649	-0.7078	1.5272	-0.9679				
1	1.2649	-0.7078	1.4960	-0.9679	0.723430	0.723710	0.041940	0.041976
2	1.2649	-0.7390	1.3397	-1.0148	0.655270	0.655800	0.012630	0.012652
3	1.2649	-0.8953	1.5272	-0.9679	0.735180	0.738360	0.023610	0.023601
4	1.1711	-0.7078	1.3397	-0.9679	0.560260	0.559440	0.011600	0.011531
5	1.2649	-0.8953	1.4960	-1.0148	0.555070	0.555790	0.014420	0.014449
6	1.2024	-0.7078	1.4335	-1.0069	0.574090	0.573520	0.022690	0.022596
7	1.2336	-0.7078	1.5272	-0.9679	0.730410	0.730270	0.042780	0.042904
8	1.0774	-0.7703	1.4960	-0.9991	0.687810	0.685610	0.011450	0.011315
9	1.1086	-0.8640	1.5272	-0.9679	0.615050	0.616180	0.011000	0.010952
10	1.2649	-0.7078	1.3710	-0.9679	0.594230	0.594310	0.024970	0.024848
11	1.2336	-0.7703	1.4960	-0.9679	0.671280	0.672420	0.030950	0.030665
12	1.0774	-0.7078	1.5272	-0.9679	0.654750	0.654770	0.025810	0.025738
13	1.0774	-0.7078	1.5272	-1.0148	0.700350	0.700070	0.020960	0.020994
14	1.1711	-0.8015	1.4647	-1.0148	0.599220	0.599570	0.011550	0.011646
15	1.2649	-0.7078	1.4022	-0.9835	0.655720	0.656140	0.027190	0.027399
16	1.1711	-0.8640	1.5272	-1.0148	0.707020	0.706820	0.012360	0.012302
17	1.2649	-0.7078	1.5272	-1.0148	0.720990	0.718590	0.041260	0.041180
18	1.0774	-0.7078	1.4335	-0.9679	0.615940	0.617320	0.013680	0.013810
19	1.1399	-0.8015	1.5272	-0.9913	0.651060	0.651240	0.018490	0.018680
20	1.1711	-0.8953	1.5272	-0.9679	0.703620	0.700210	0.013810	0.013873
21	1.2649	-0.8640	1.4022	-0.9679	0.705010	0.707330	0.011090	0.011271
22	1.2649	-0.8640	1.5272	-1.0148	0.733140	0.732410	0.021830	0.021892
23	1.2649	-0.8953	1.4335	-0.9913	0.547420	0.546450	0.009580	0.009376
opt	1.1369	-0.8541	1.5272	-1.0148	0.624890	0.623320	0.010310	0.010000

Table 11.10: Transonic airfoil design response surface cycle 4 results

CYCLE 5

	ξ1	ξ2	ξ3	ξ4	CL	Fit	CD	Fit
min	1.1134	-0.8775	1.4335	-1.0381				
max	1.1603	-0.8307	1.6210	-0.9913				
1	1.1134	-0.8307	1.6210	-1.0303	0.676670	0.674970	0.021040	0.020810
2	1.1603	-0.8307	1.6210	-1.0147	0.687820	0.688640	0.027100	0.027318
3	1.1369	-0.8775	1.6210	-1.0147	0.662500	0.662710	0.019430	0.019413
4	1.1134	-0.8541	1.5272	-0.9913	0.620970	0.620720	0.010540	0.010438
5	1.1212	-0.8307	1.6210	-0.9913	0.679910	0.680970	0.025690	0.025785
6	1.1134	-0.8307	1.5272	-0.9913	0.634490	0.634910	0.012740	0.012803
7	1.1134	-0.8463	1.6210	-1.0303	0.666240	0.667190	0.018900	0.019047
8	1.1603	-0.8775	1.5897	-1.0303	0.651780	0.651820	0.015830	0.015938
9	1.1603	-0.8697	1.6210	-0.9913	0.675790	0.676090	0.025180	0.025275
10	1.1290	-0.8463	1.5585	-0.9913	0.649830	0.649080	0.016610	0.016617
11	1.1525	-0.8307	1.4960	-1.0147	0.627120	0.626380	0.010780	0.010725
12	1.1525	-0.8307	1.6210	-0.9913	0.688480	0.687250	0.029320	0.029135
13	1.1525	-0.8619	1.5272	-1.0147	0.625780	0.626130	0.011080	0.011071
14	1.1134	-0.8775	1.6210	-0.9913	0.659100	0.658860	0.019440	0.019443
15	1.1603	-0.8307	1.4960	-0.9913	0.636510	0.637040	0.013630	0.013705
16	1.1603	-0.8697	1.6210	-1.0381	0.670750	0.670030	0.020280	0.020136
17	1.1447	-0.8307	1.6210	-1.0381	0.682290	0.682820	0.023220	0.023263
18	1.1447	-0.8775	1.5272	-0.9913	0.620900	0.620920	0.010970	0.010989
19	1.1212	-0.8307	1.5585	-1.0381	0.641770	0.642210	0.012750	0.012826
20	1.1212	-0.8775	1.5897	-1.0381	0.632070	0.632150	0.011450	0.011451
21	1.1603	-0.8775	1.6210	-0.9913	0.673150	0.673220	0.024370	0.024303
22	1.1603	-0.8307	1.5585	-1.0147	0.663290	0.663060	0.019060	0.018909
23	1.1603	-0.8307	1.5272	-1.0381	0.640460	0.640530	0.012730	0.012740
opt	1.6030	-0.8775	1.5330	-1.0216	0.620660	0.621278	0.010150	0.010000

Table 11.11: Transonic airfoil design response surface cycle 5 results

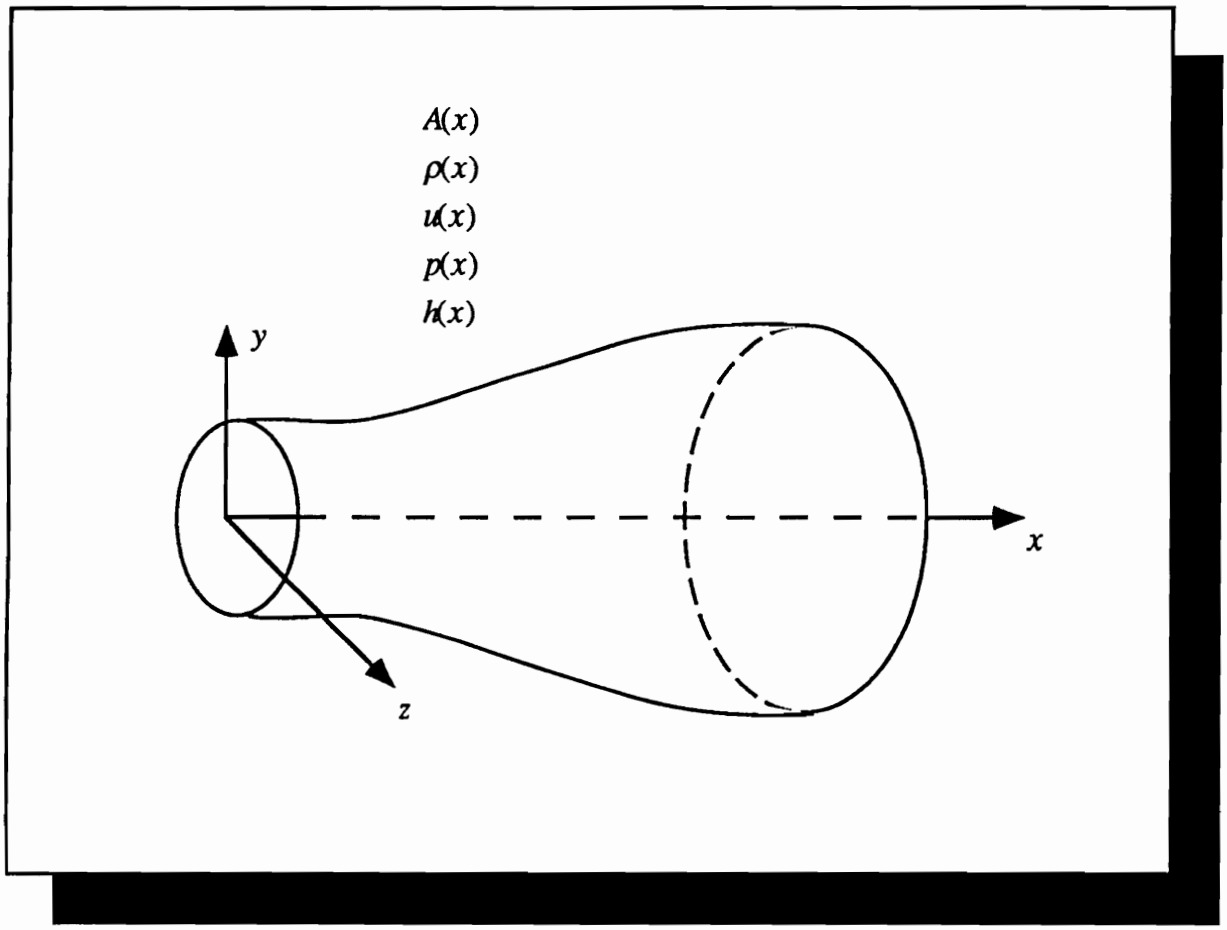

Figure 1.1: Application of quasi-one-dimensional flow theory.

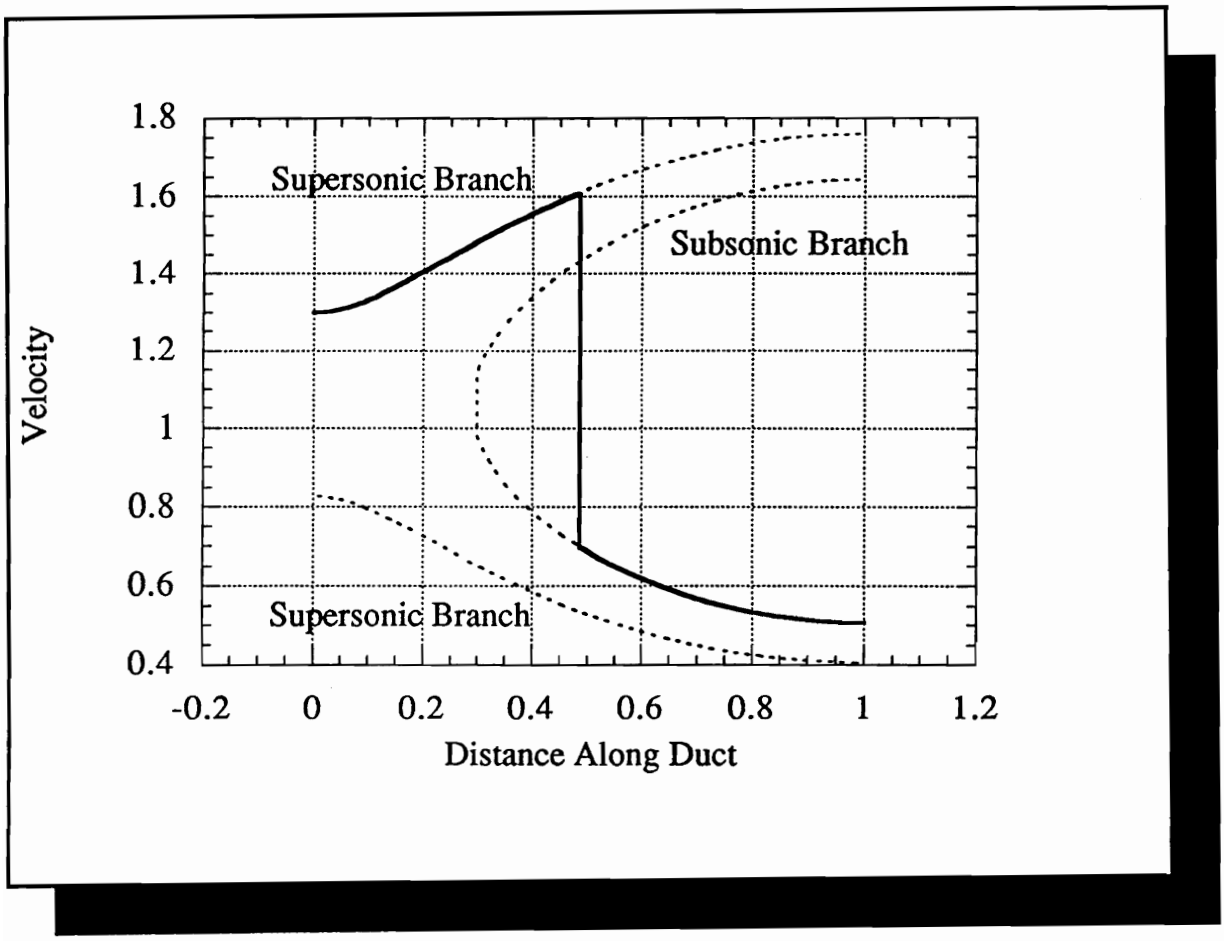

Figure 1.2: Supersonic and subsonic branches of the exact solution to $f_x+g=0$.

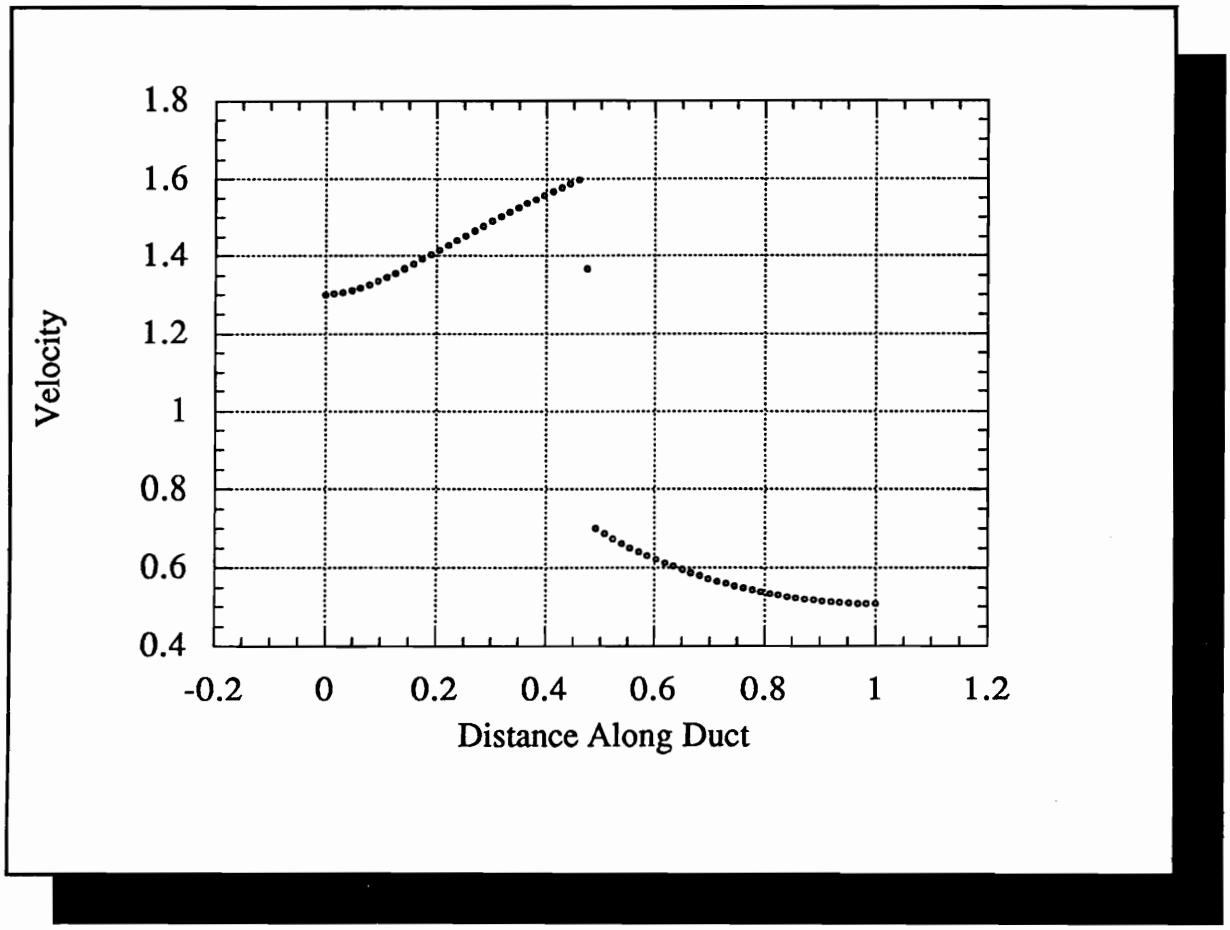

Figure 1.3: Godunov solution to $f_x+g=0$ computed on a 64 point grid.

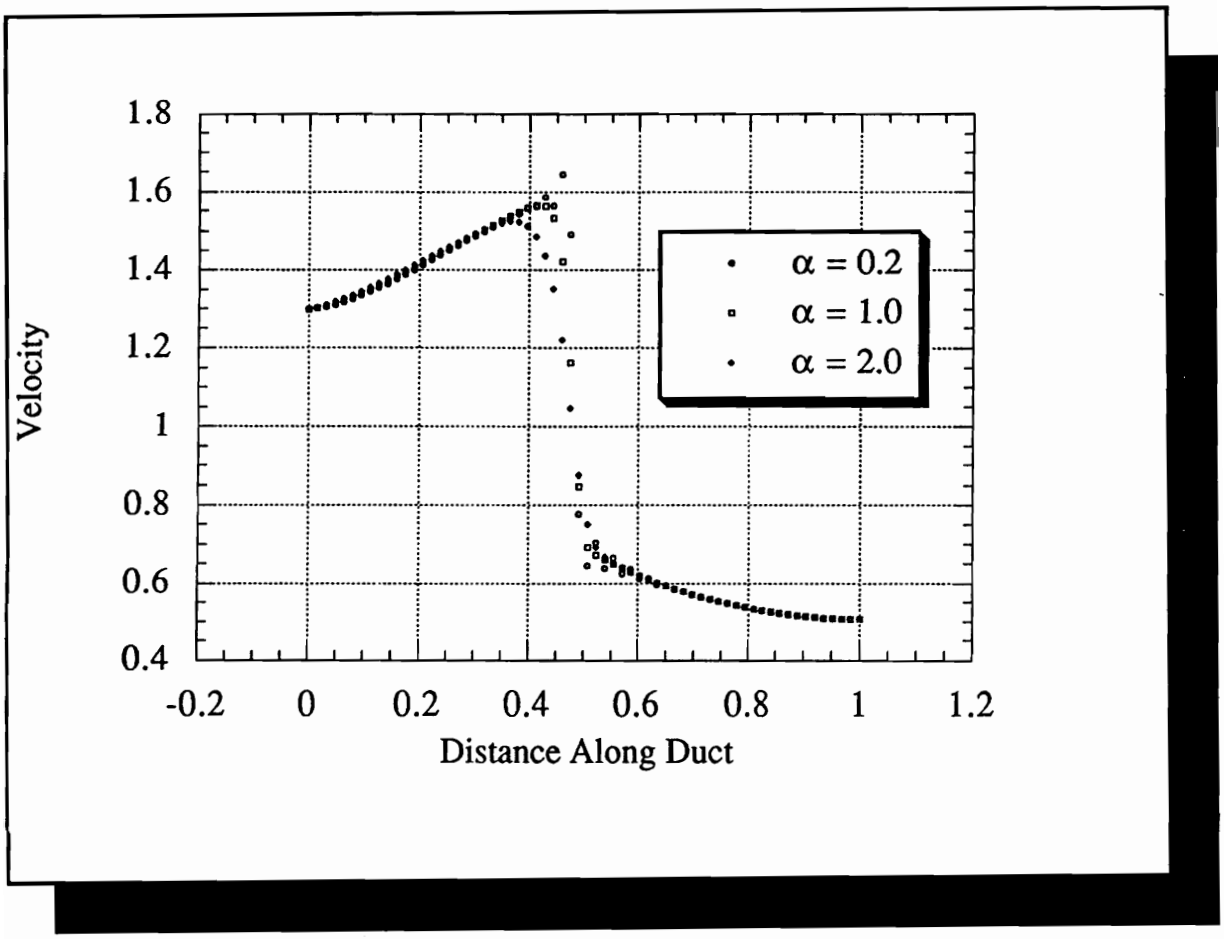

Figure 1.4: Artificial viscosity solution to $f_x+g=0$ computed on a 64 point grid with several values of α.

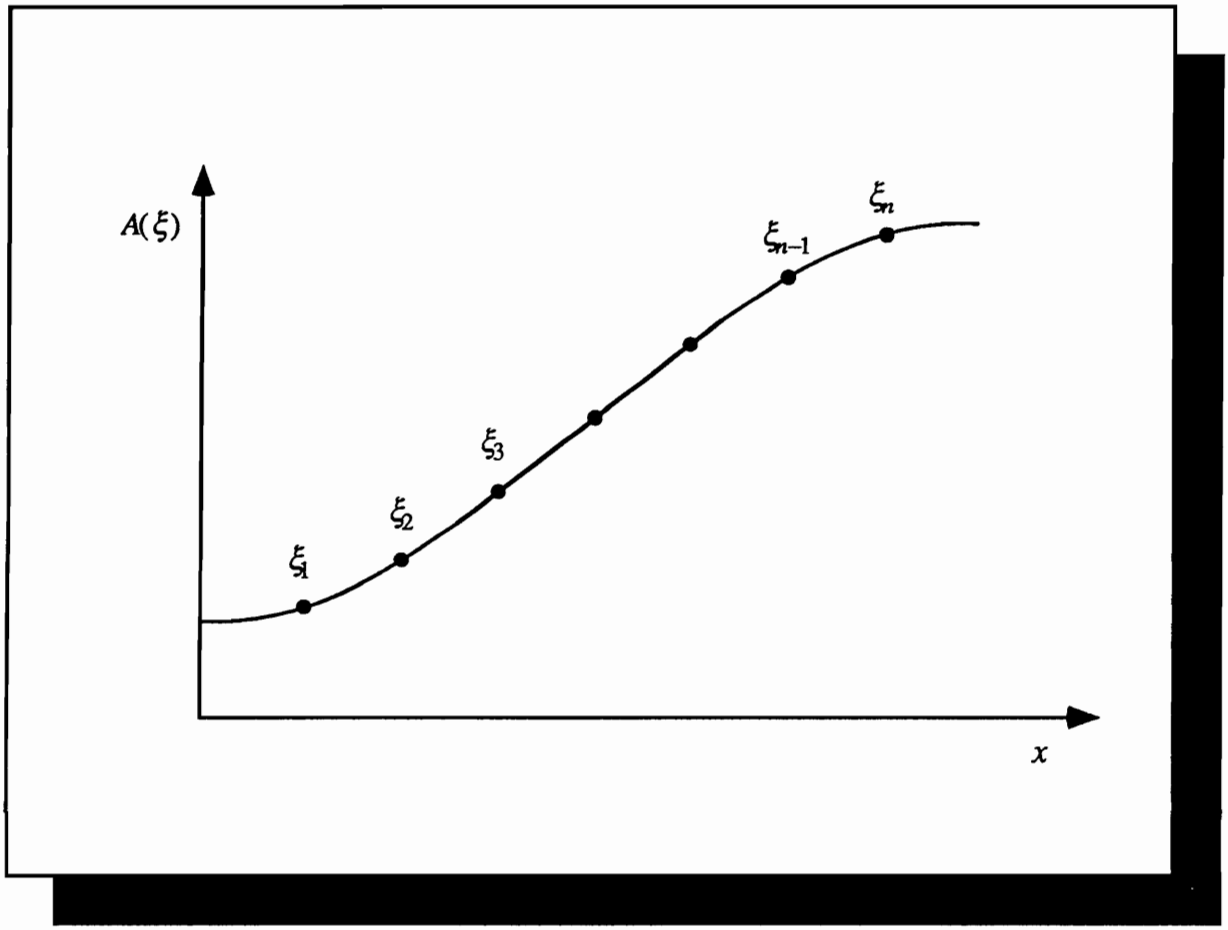

Figure 1.5: Design variable parameterization of the quasi-one-dimensional duct

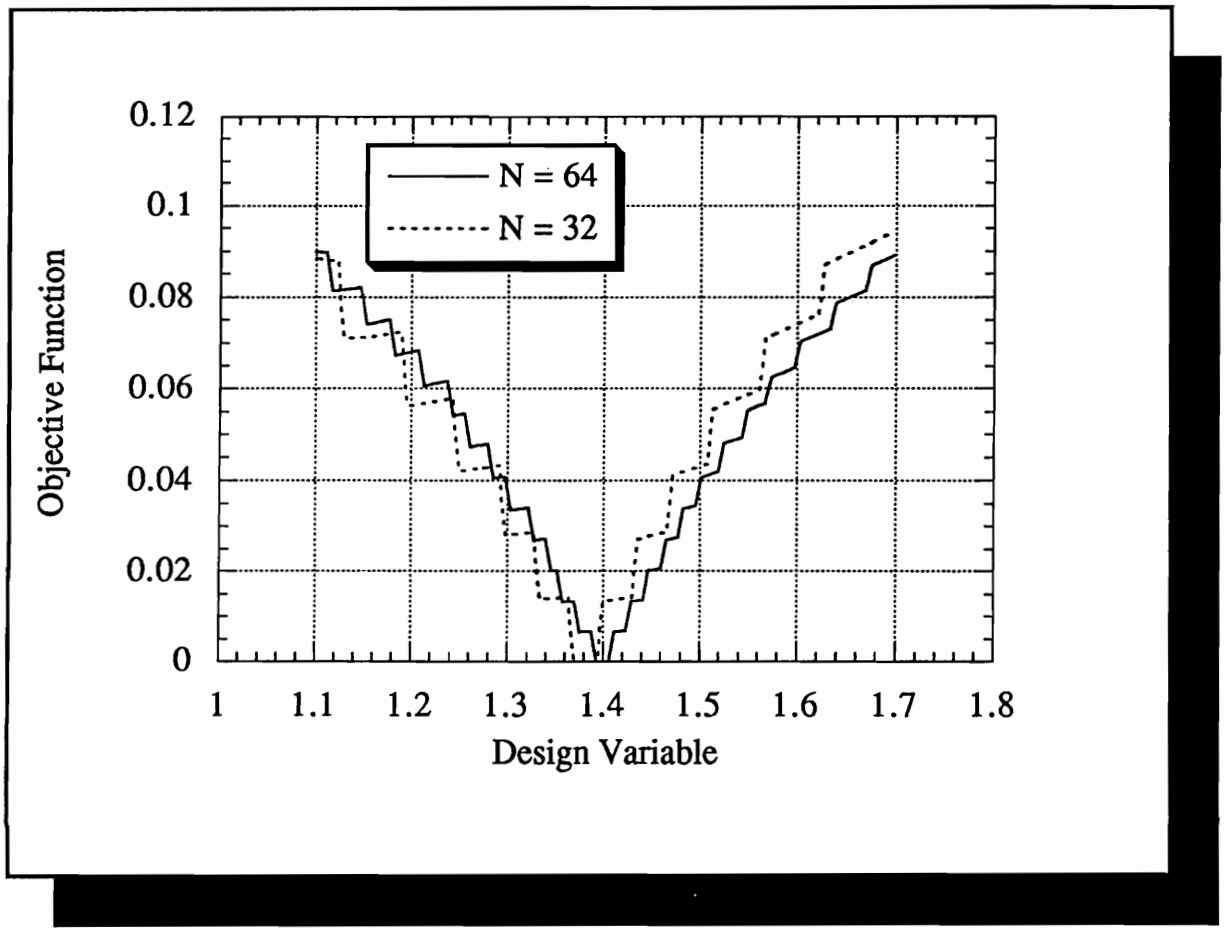

Figure 1.6: Discontinuous objective function for the univariate case using the exact flow solution and $N = 32$ and $N = 64$.

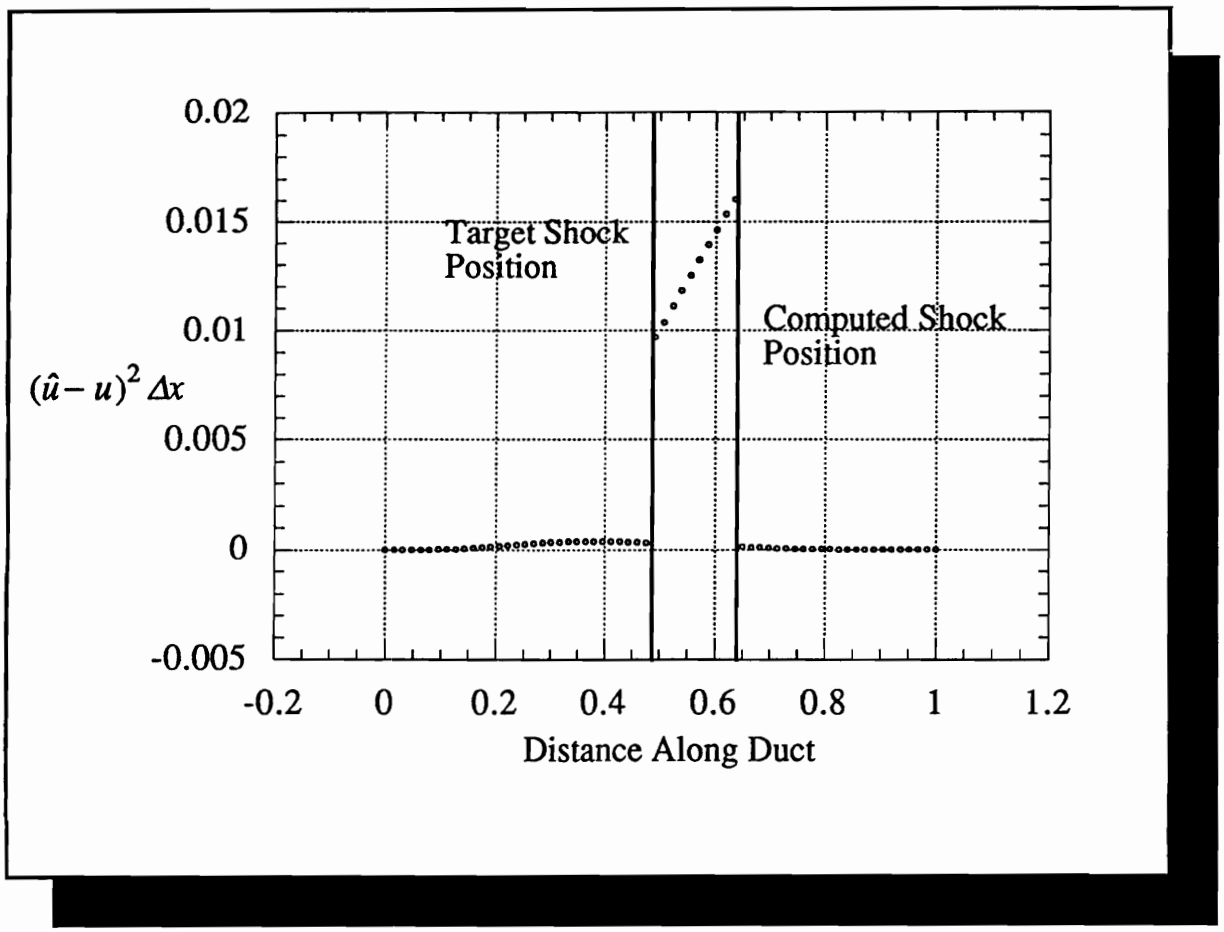

Figure 1.7: Plot of terms in summation (1.24) reveals that terms between shocks dominate the summation.

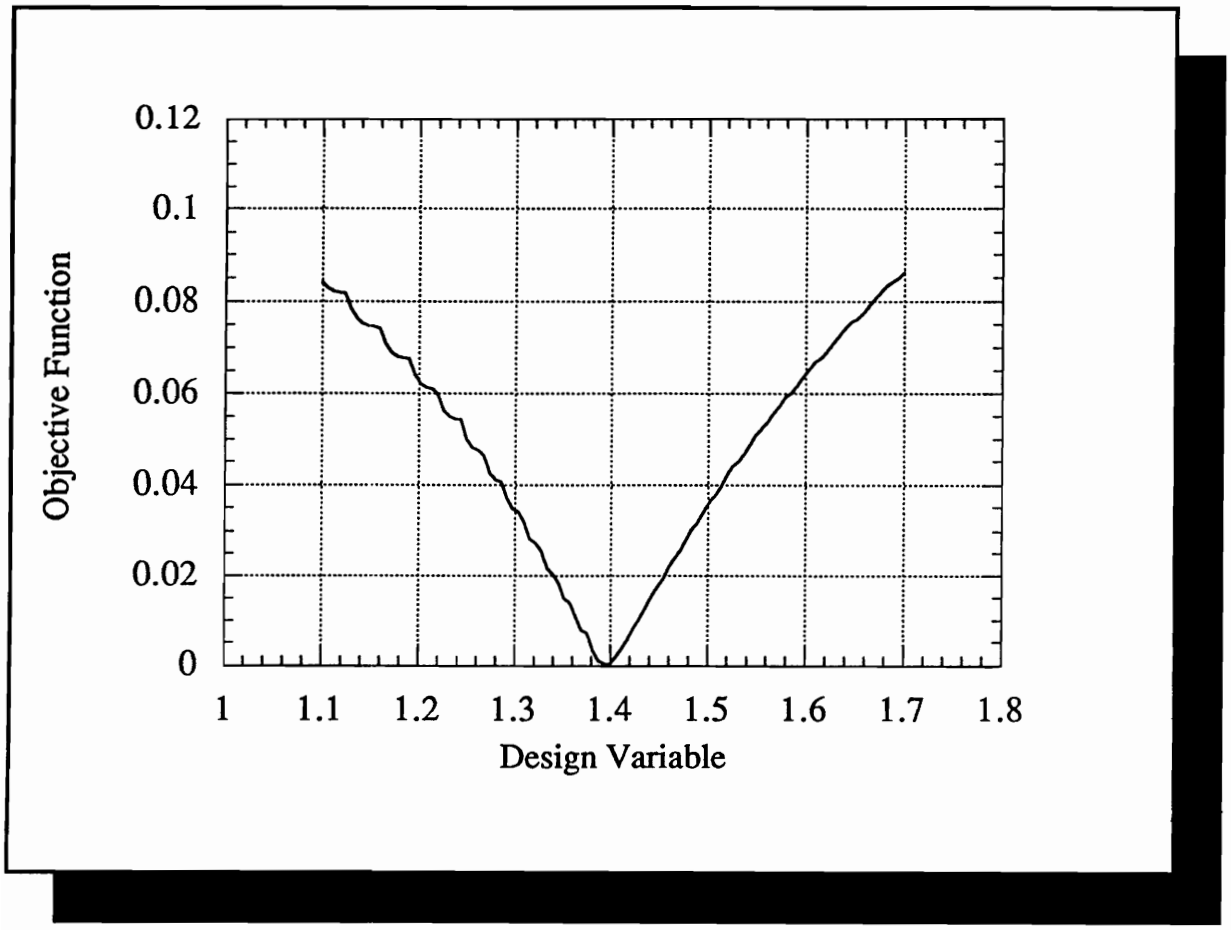

Figure 1.8: Non-smooth objective function for the univariate case using the Godunov flow solver for $N = 64$.

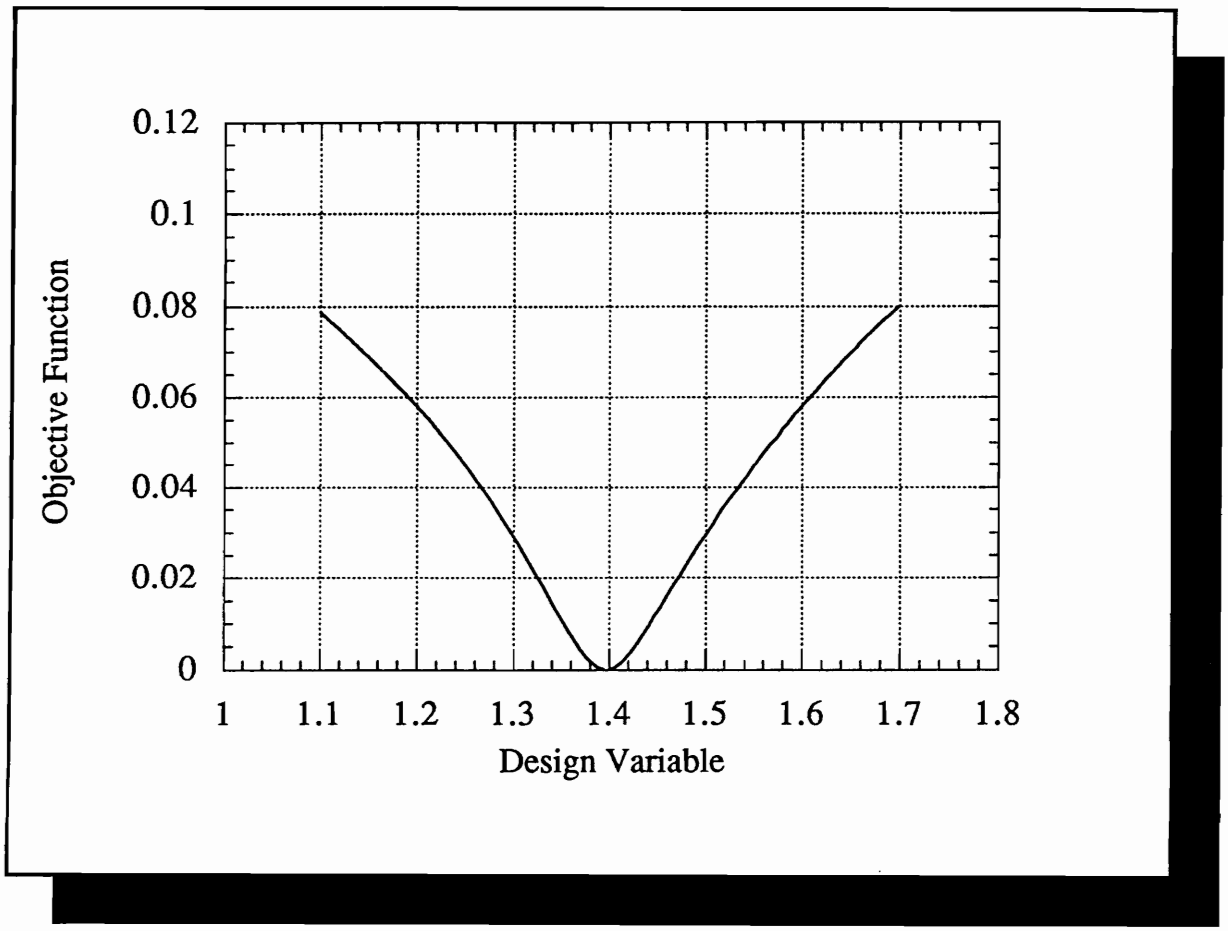

Figure 1.9: Non-smooth objective function for the univariate case using the artificial viscosity flow solver for $N = 64$.

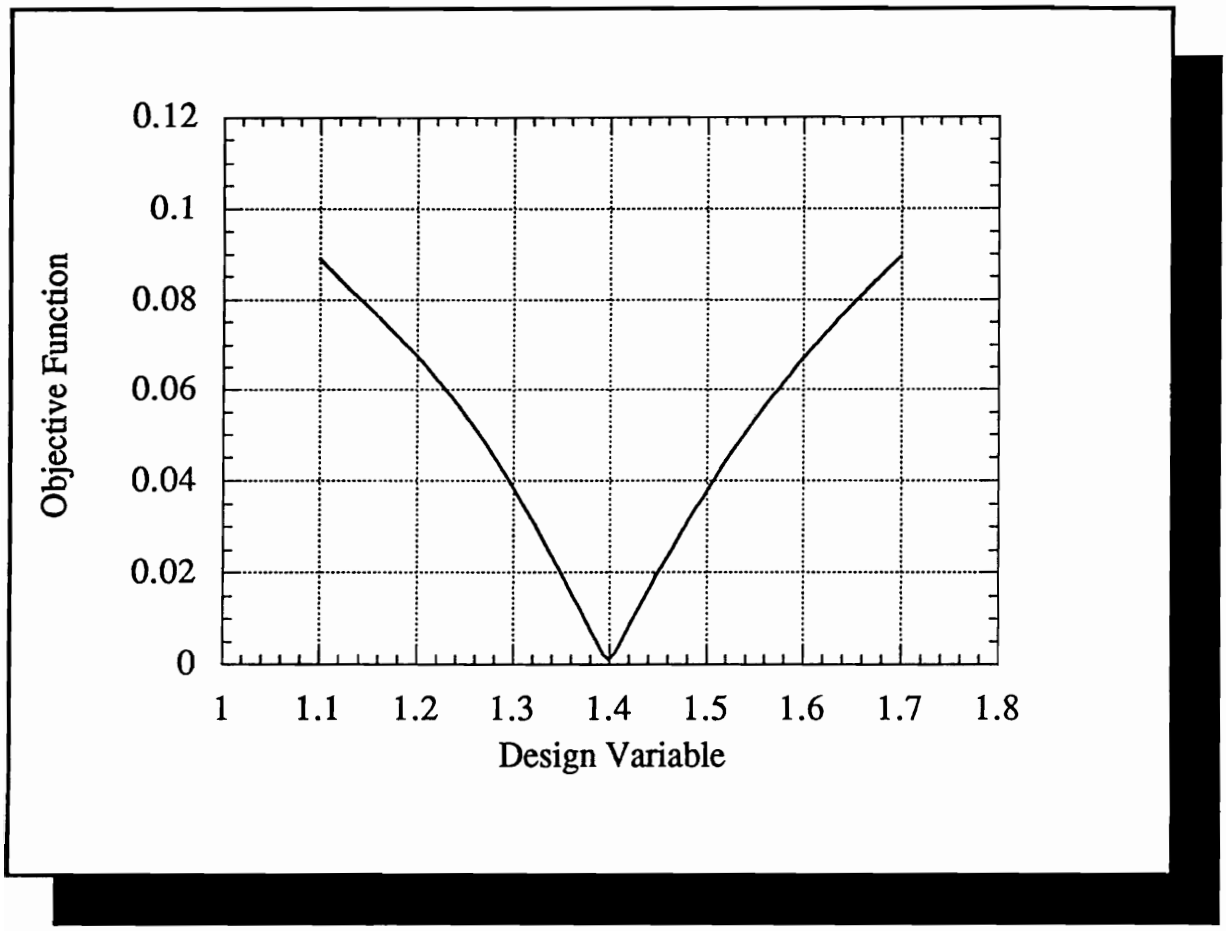

Figure 1.10: Shock-fitted objective function for the univariate case using the exact flow solver for $N = 64$.

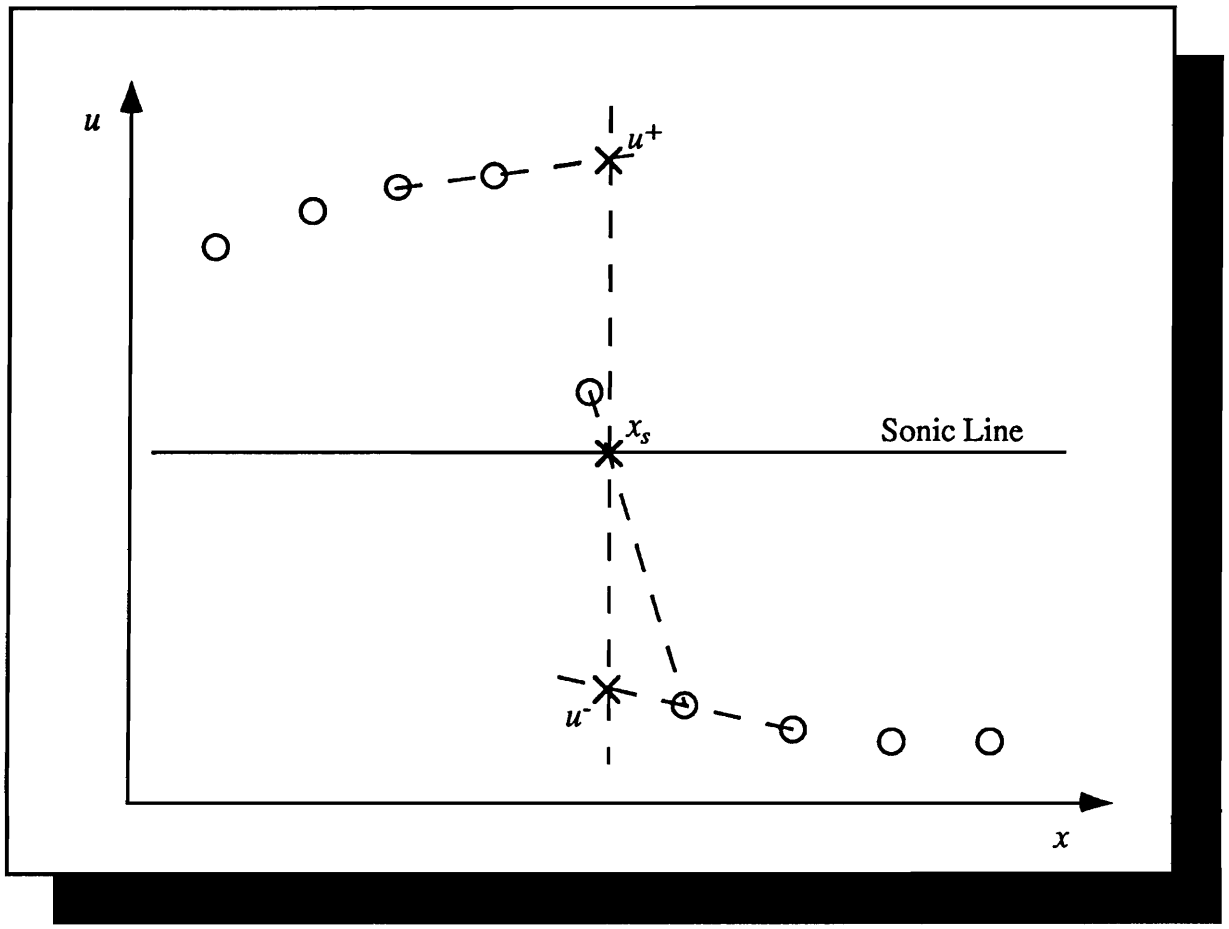

Figure 1.11: Schematic of interpolating the shock position and extrapolating left and right velocities at the shock.

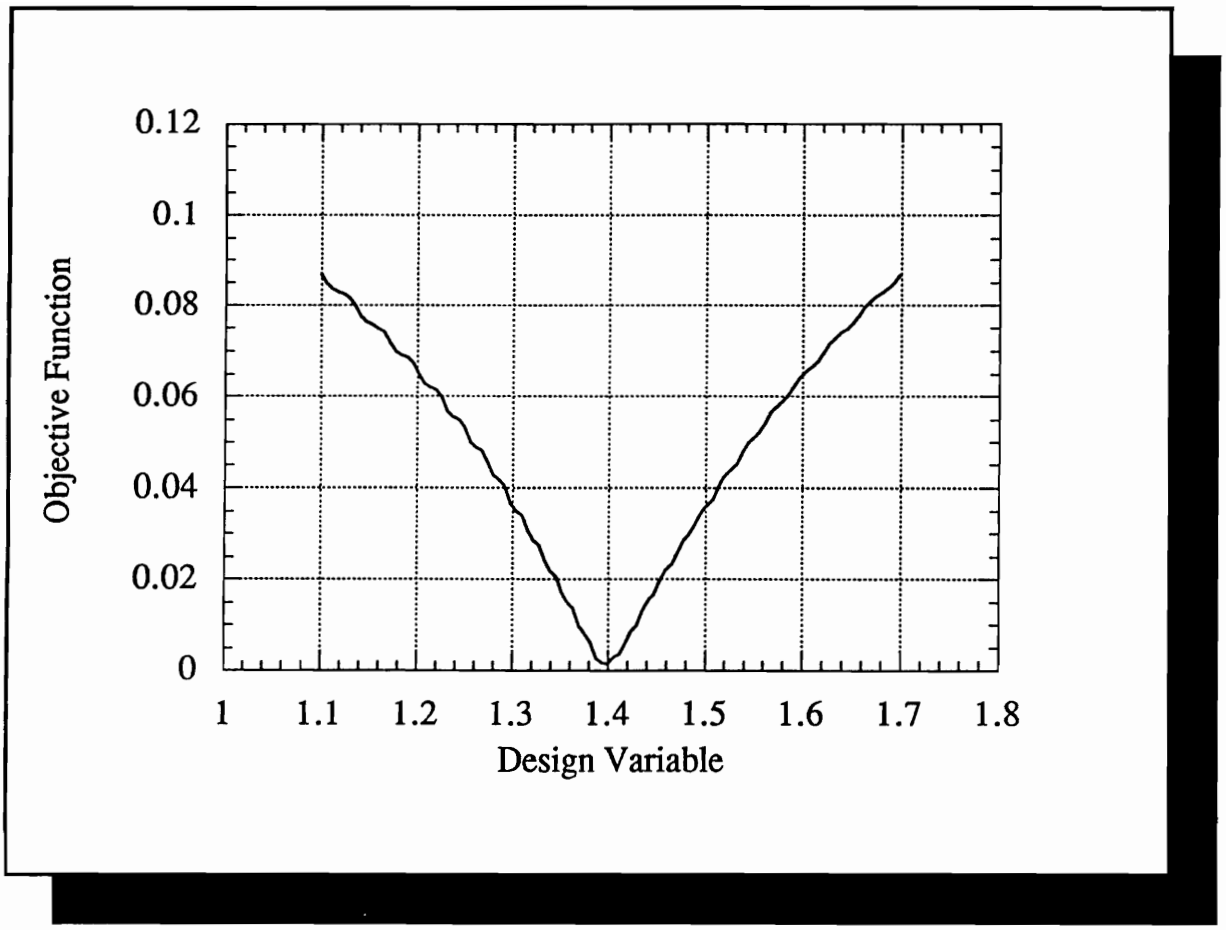

Figure 1.12: Shock-fitted objective function for the univariate case using the Godunov flow solver for $N = 64$.

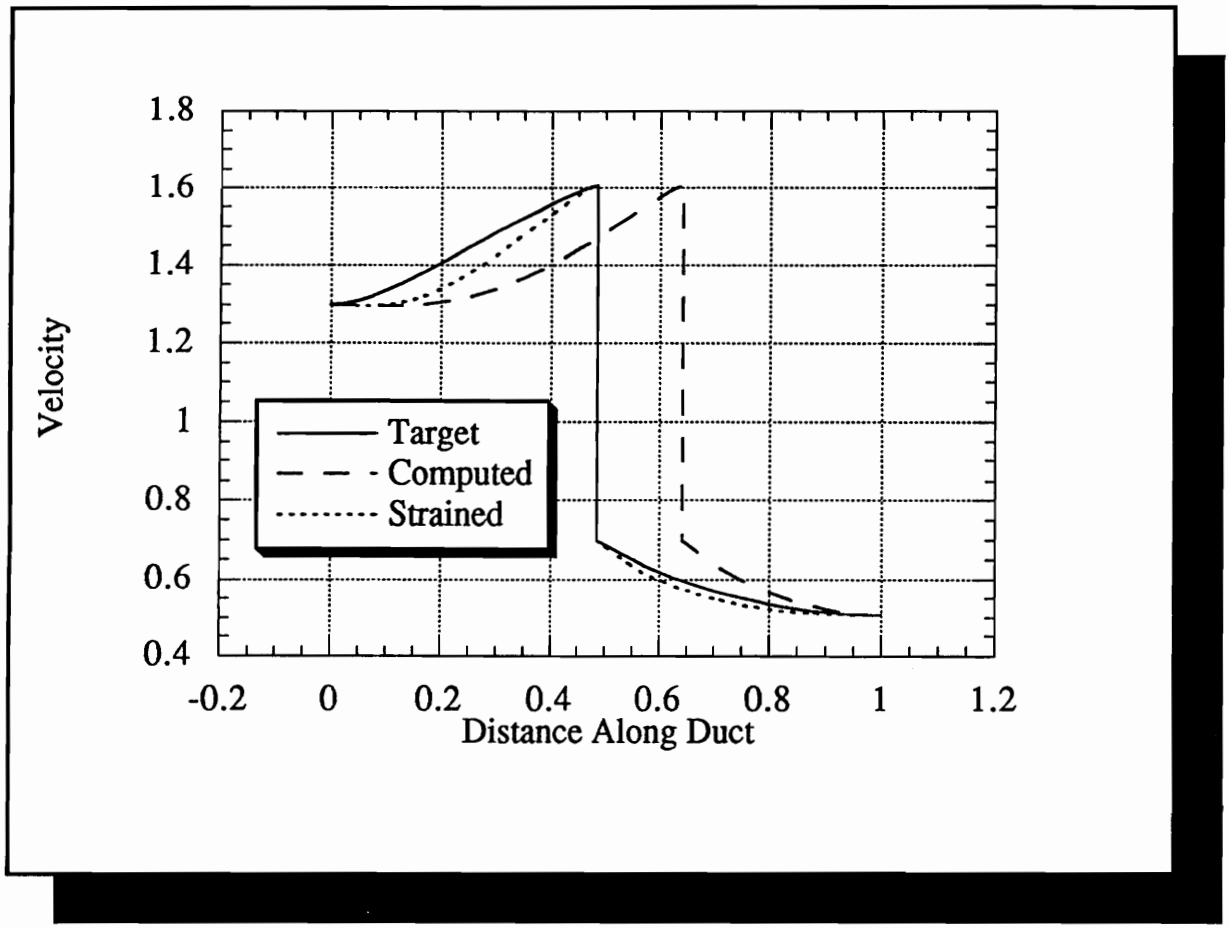

Figure 1.13: Coordinate straining performed for a test case computed from an exact solution.

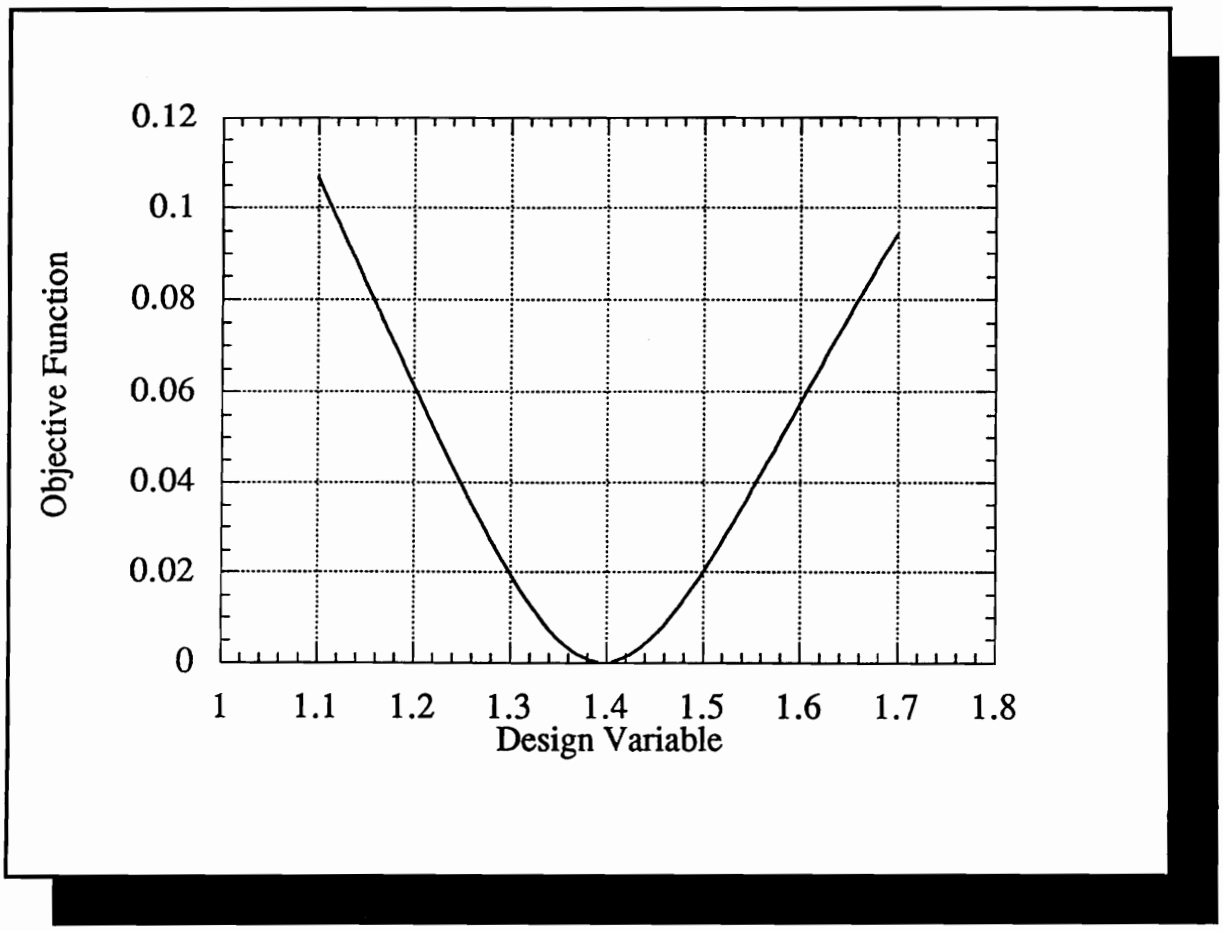

Figure 1.14: Coordinate-strained objective function for the univariate case using the exact flow solver for $N = 64$.

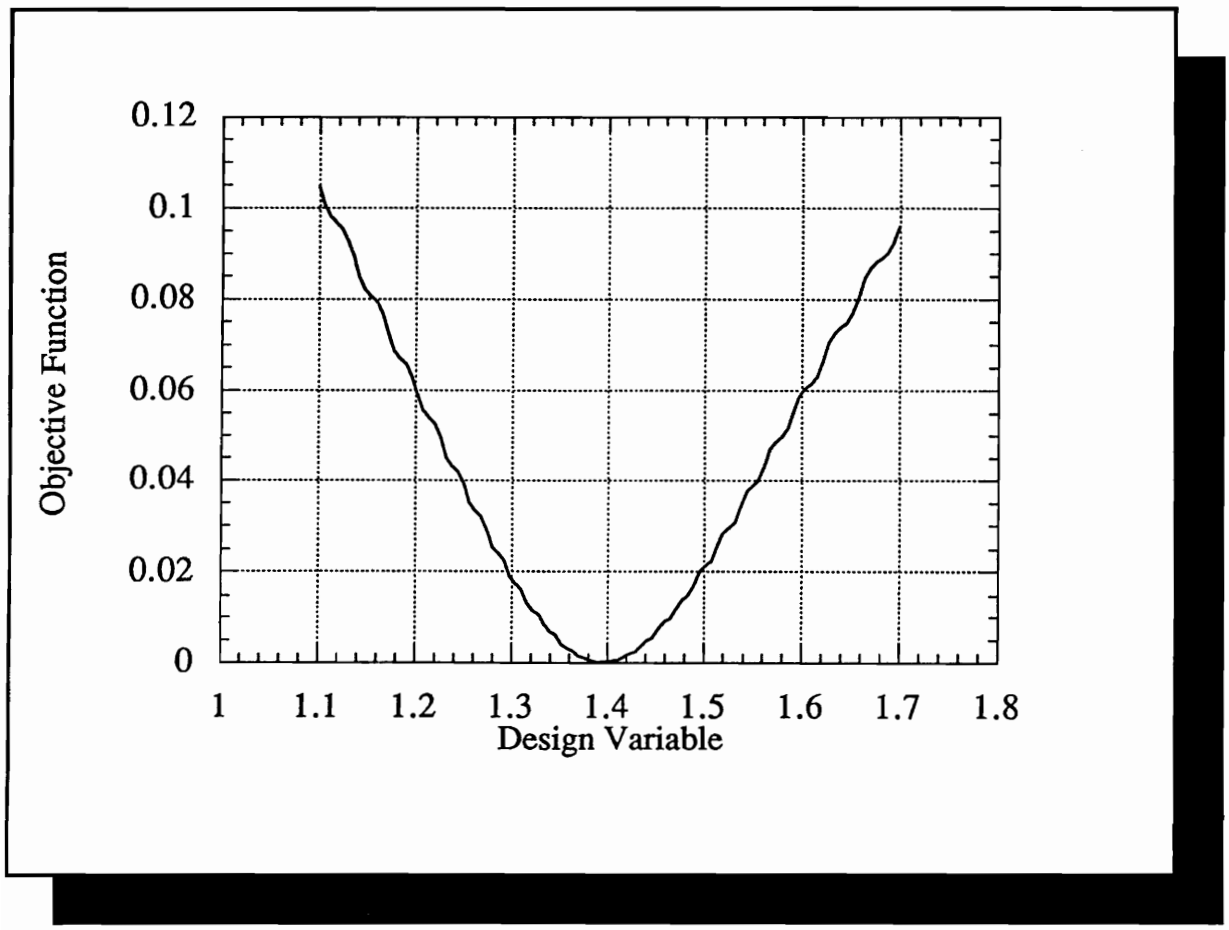

Figure 1.15: Coordinate-strained objective function for the univariate case using the Godunov flow solver for $N = 64$.

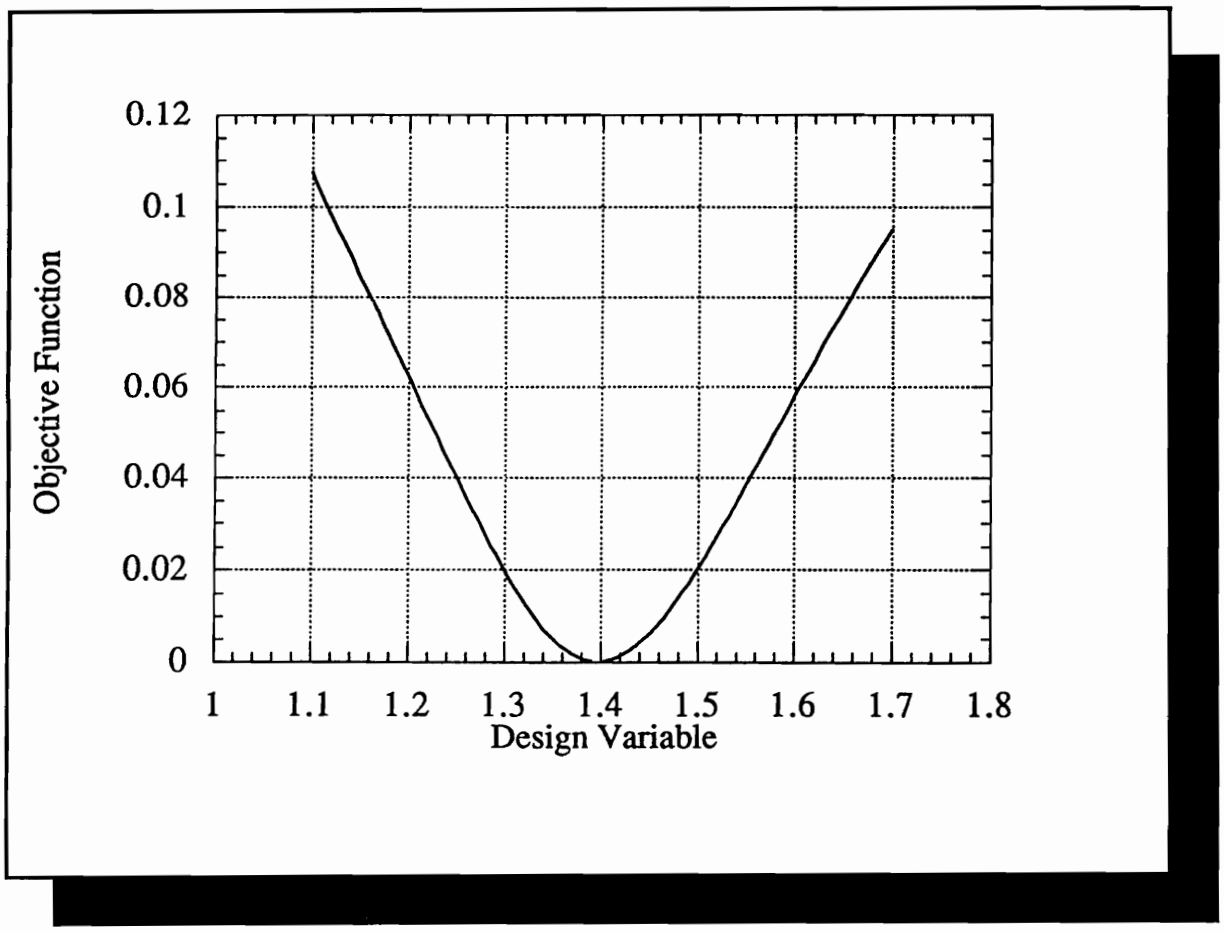

Figure 1.16: Coordinate-strained objective function for the univariate case using the artificial viscosity flow solver for $N = 64$.

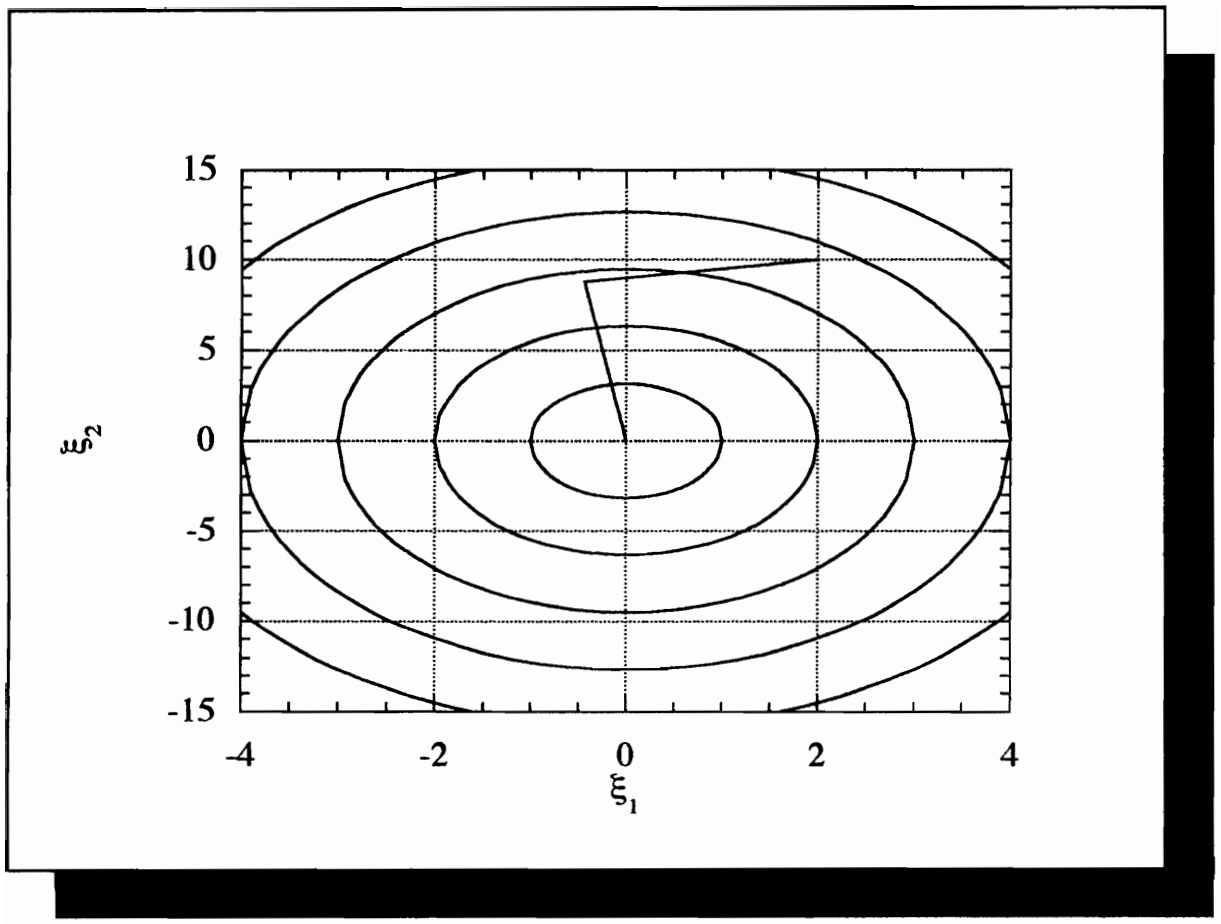

Figure 2.2: Conjugate gradient optimization of a quadratic in two variables.

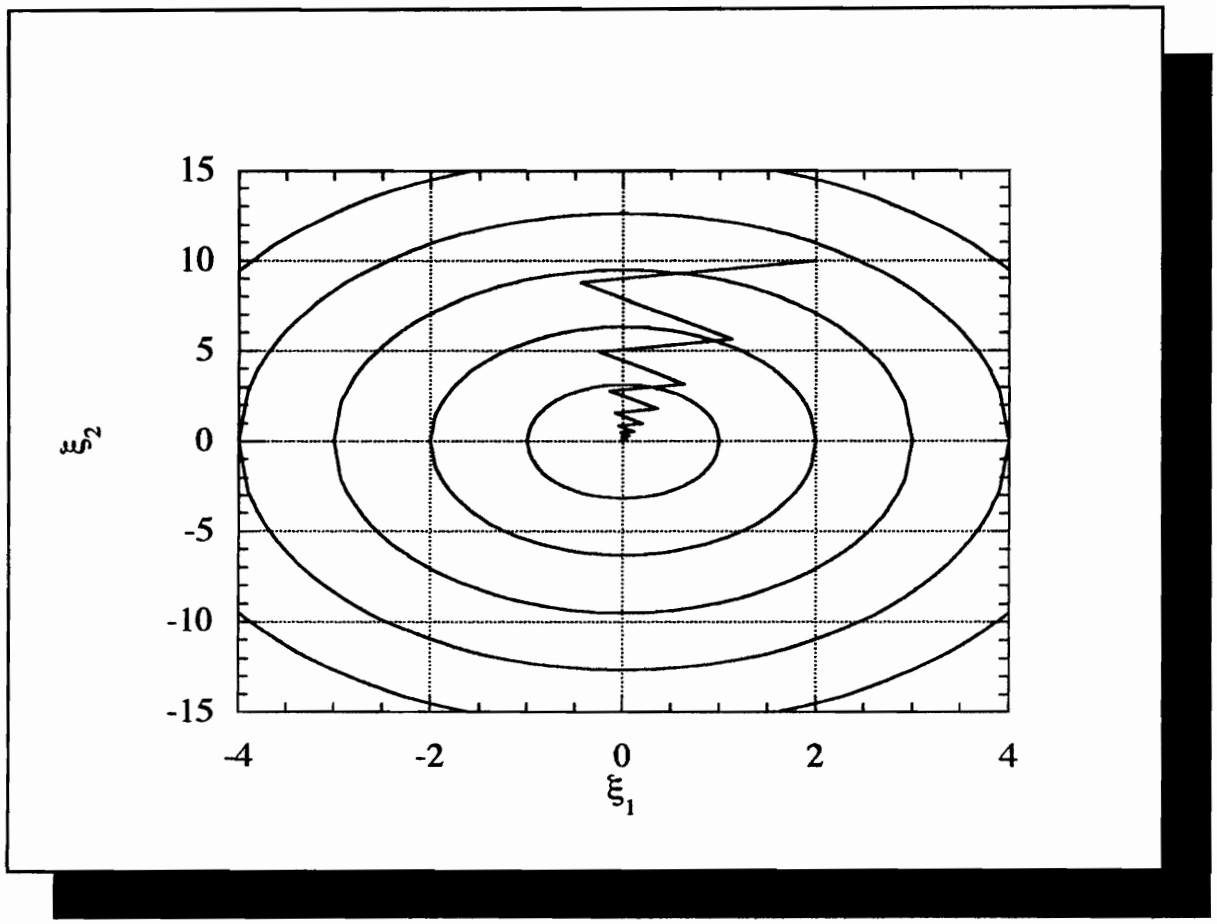

Figure 2.1: Steepest descent optimization of a quadratic in two variables.

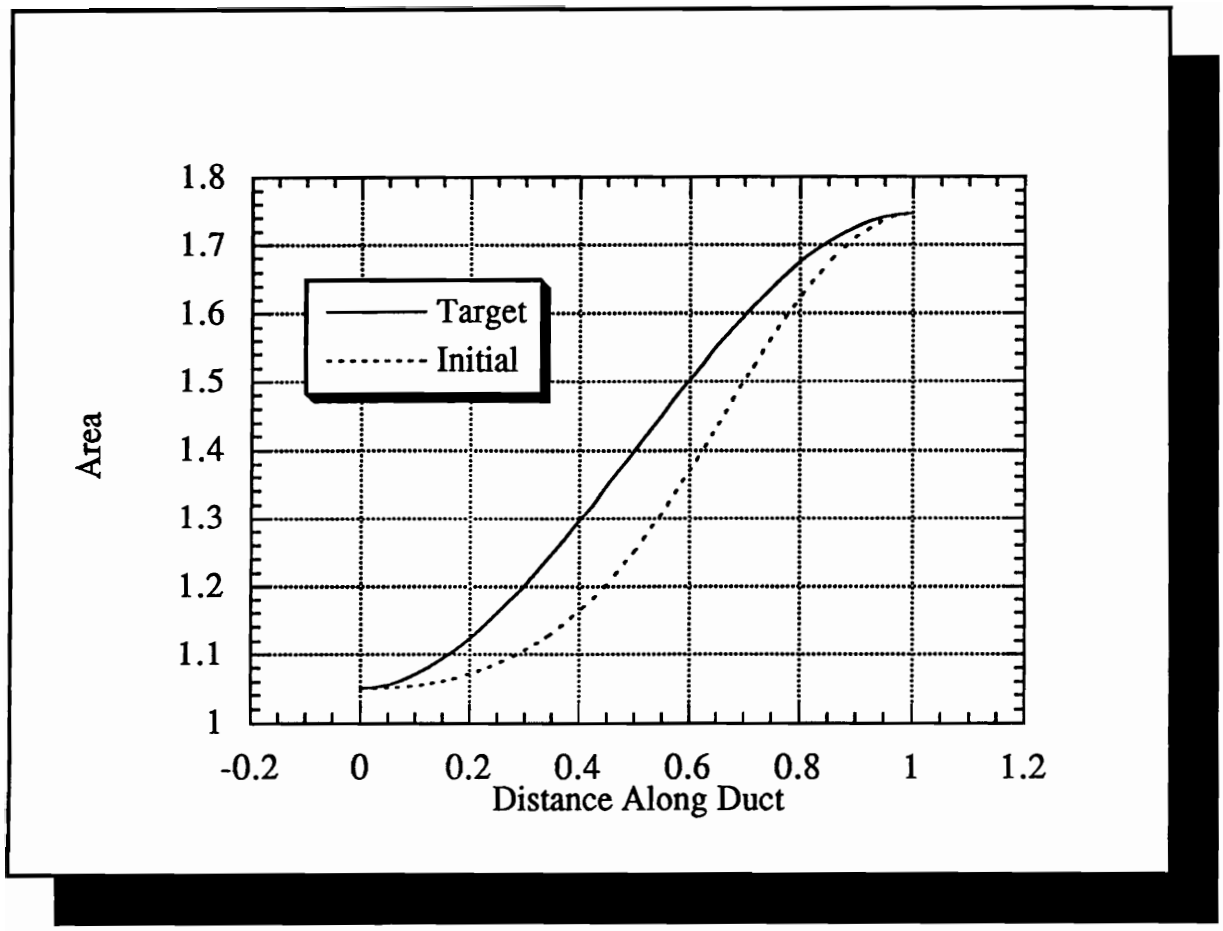

Figure 5.1: Initial and Target area distribution.

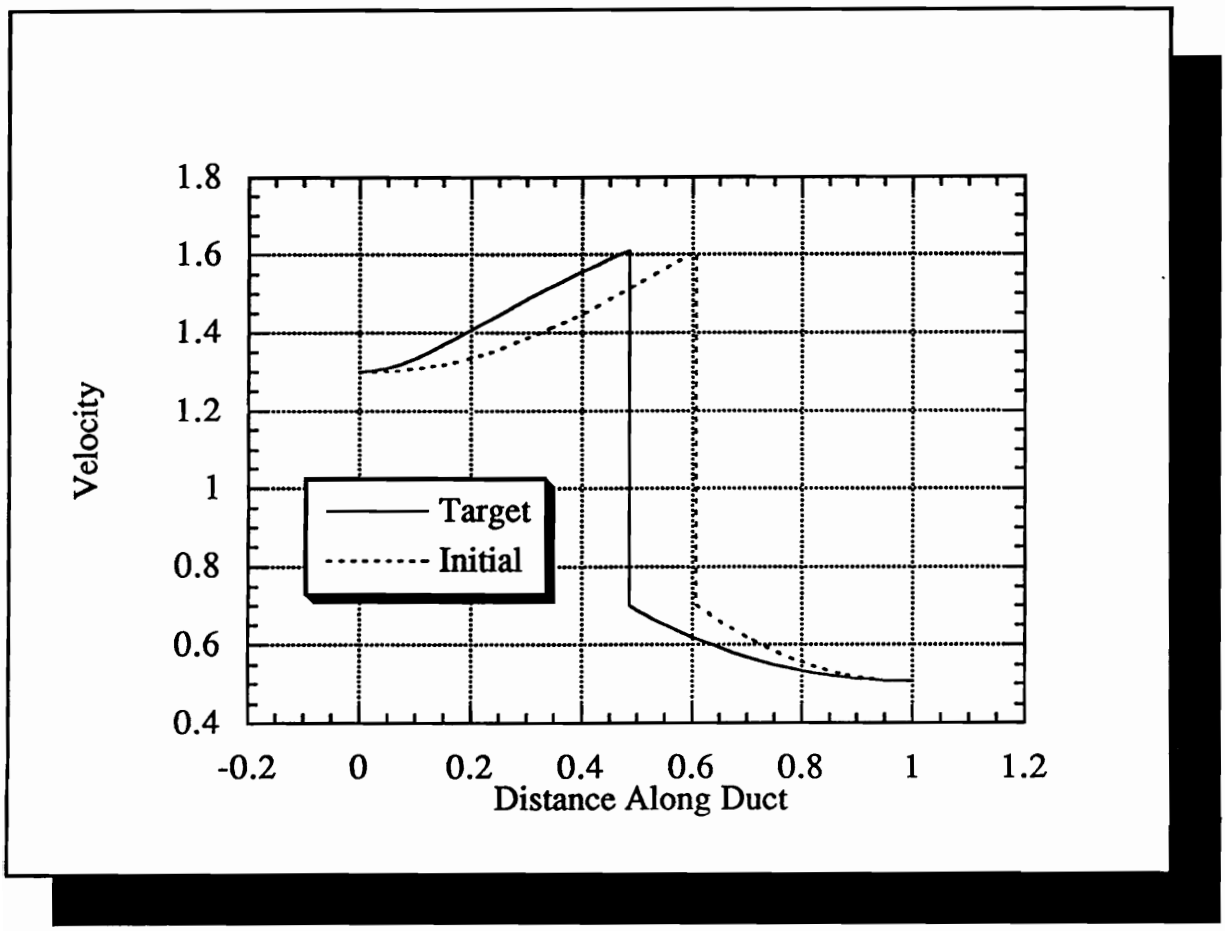

Figure 5.2: Initial and target velocity profiles using the exact flow solver.

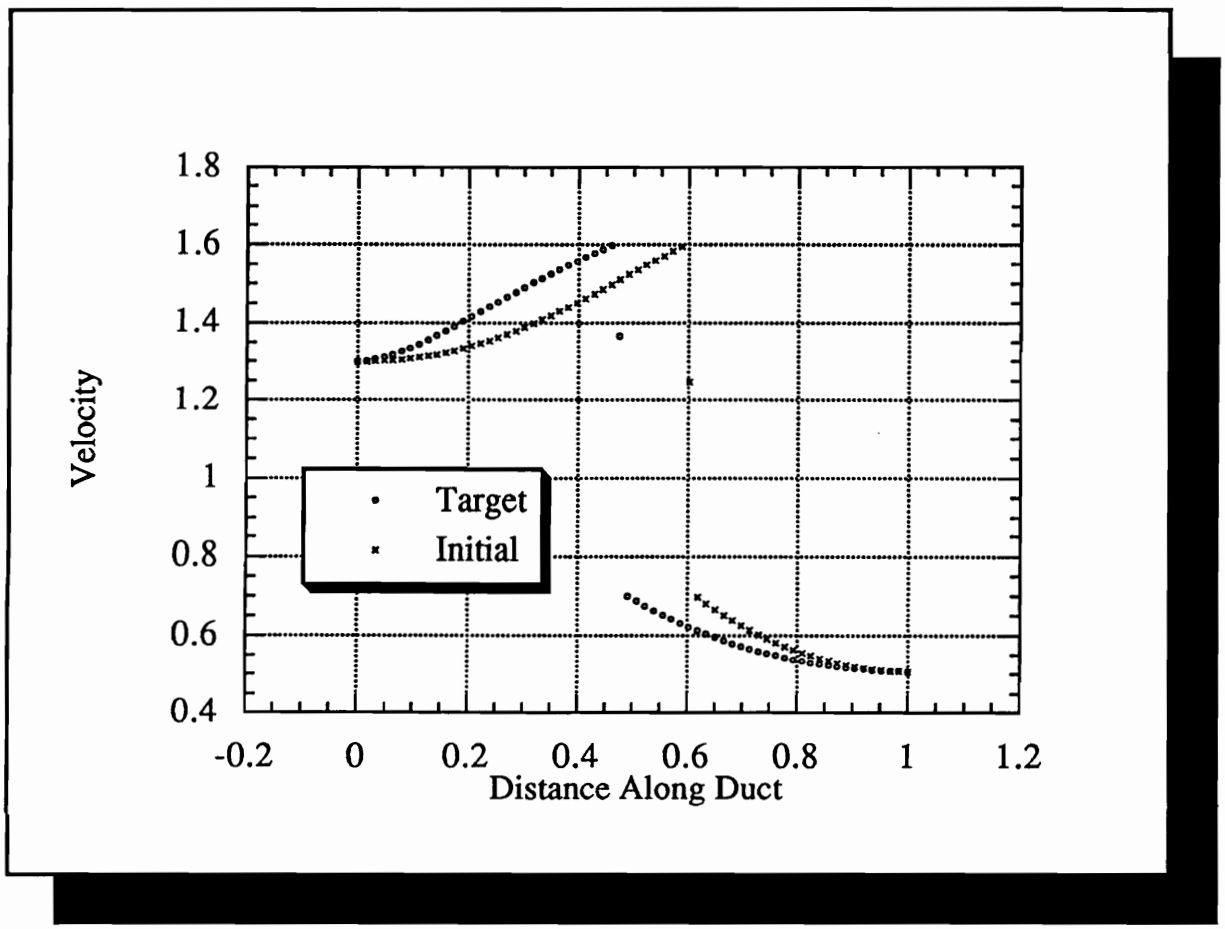

Figure 5.3: Initial and target velocity profiles using the Godunov flow solver.

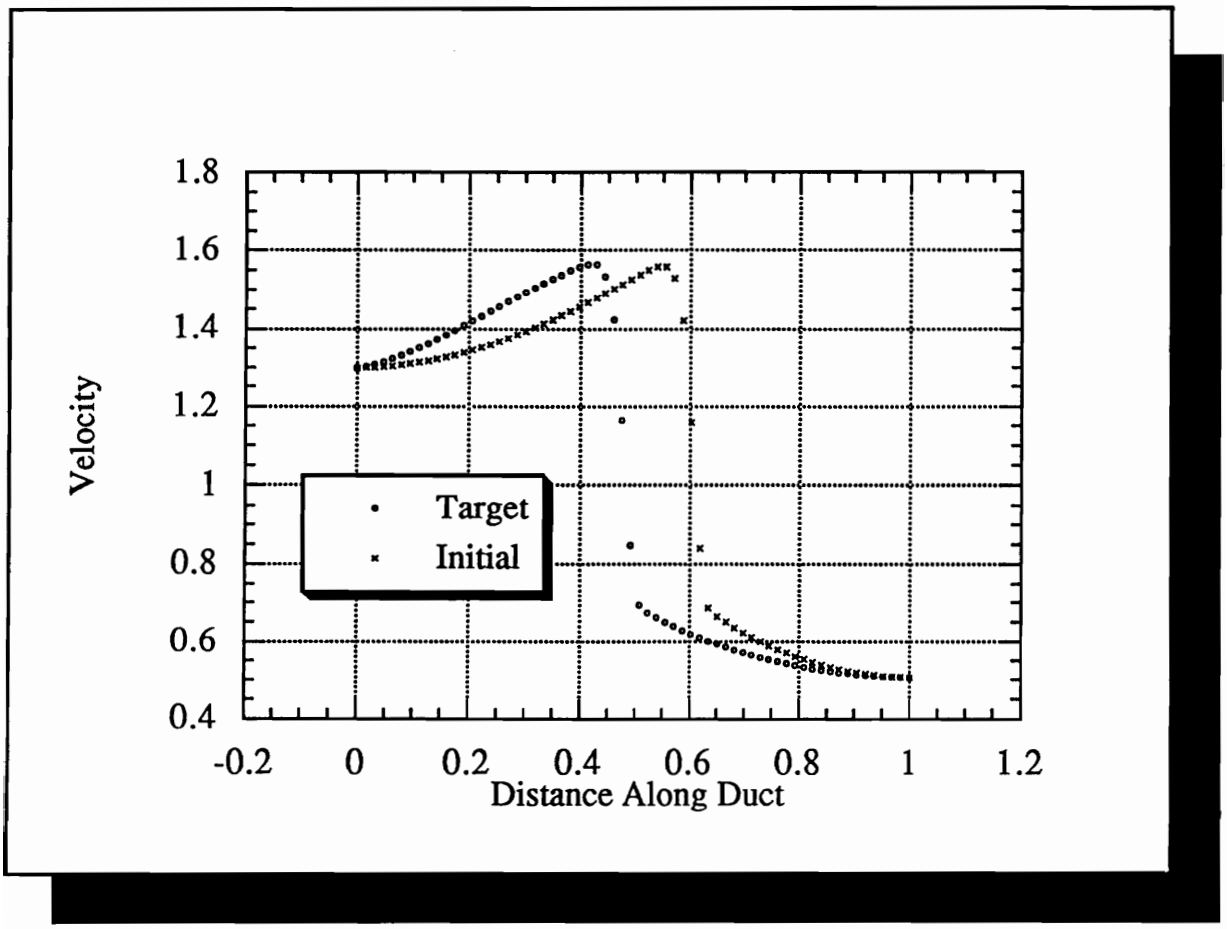

Figure 5.4: Initial and target velocity profiles using the artificial viscosity flow solver.

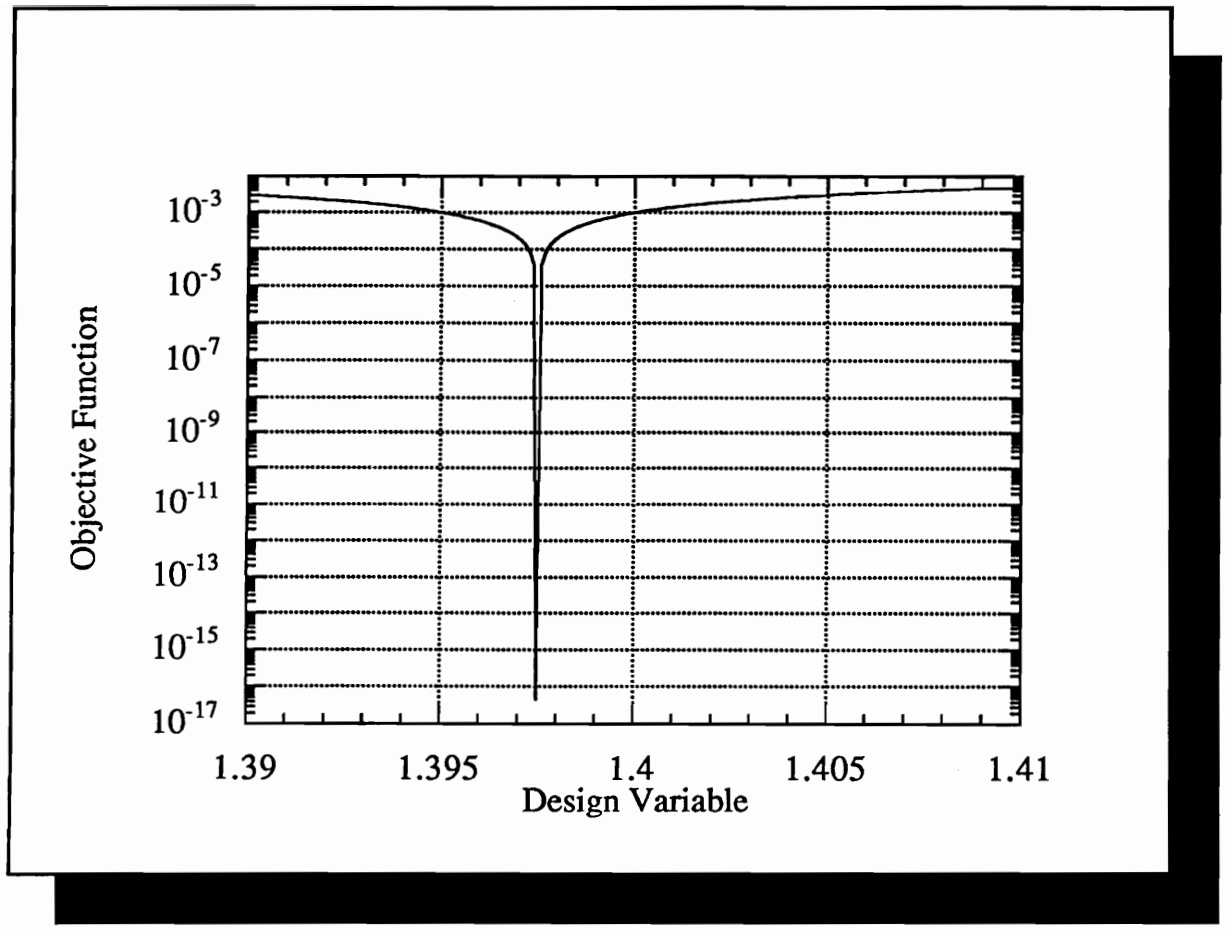

Figure 5.5: Expanded view of the shock-fitted objective function near the minimum. Figure drawn with the exact flow solutions.

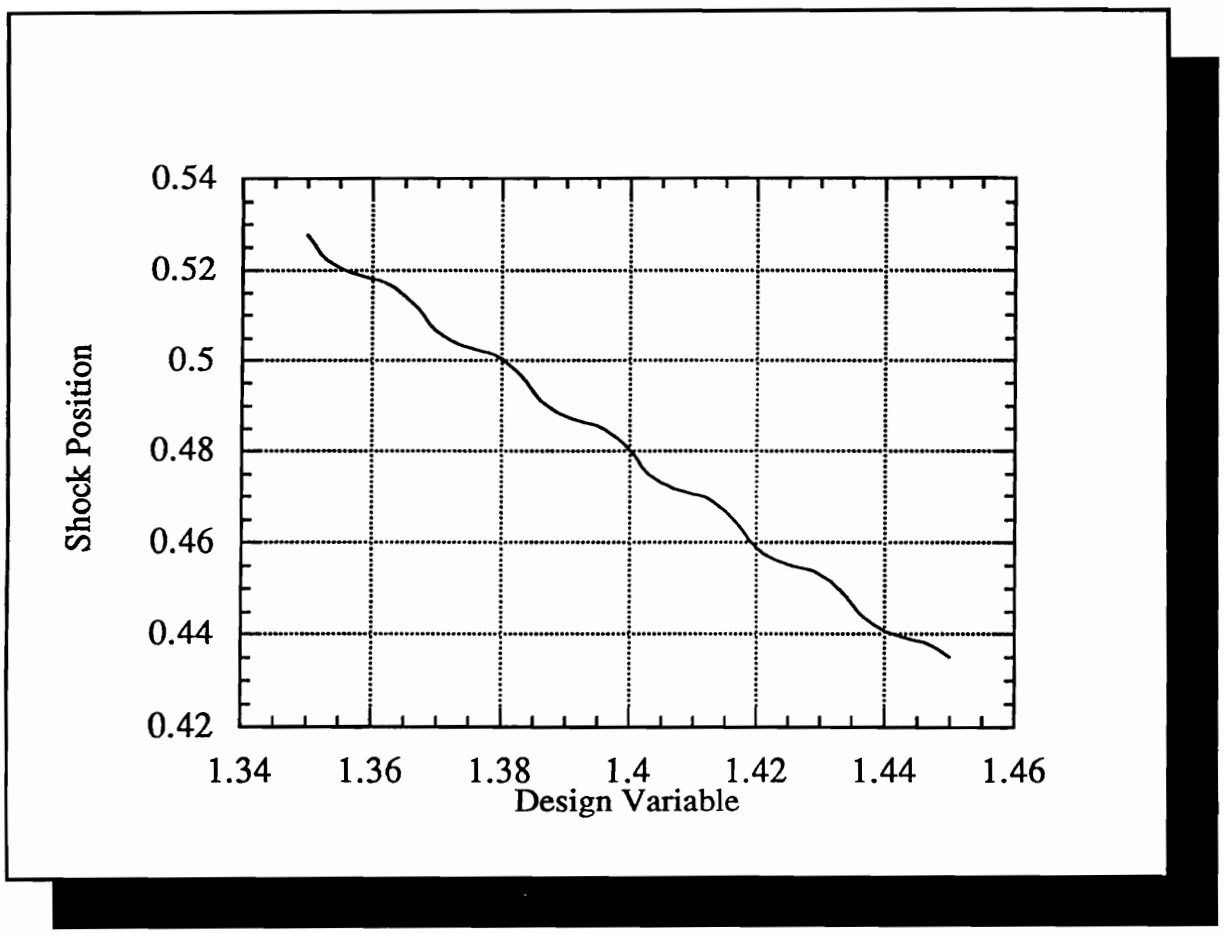

Figure 5.6: Shock position variation with design variable (univariate case).

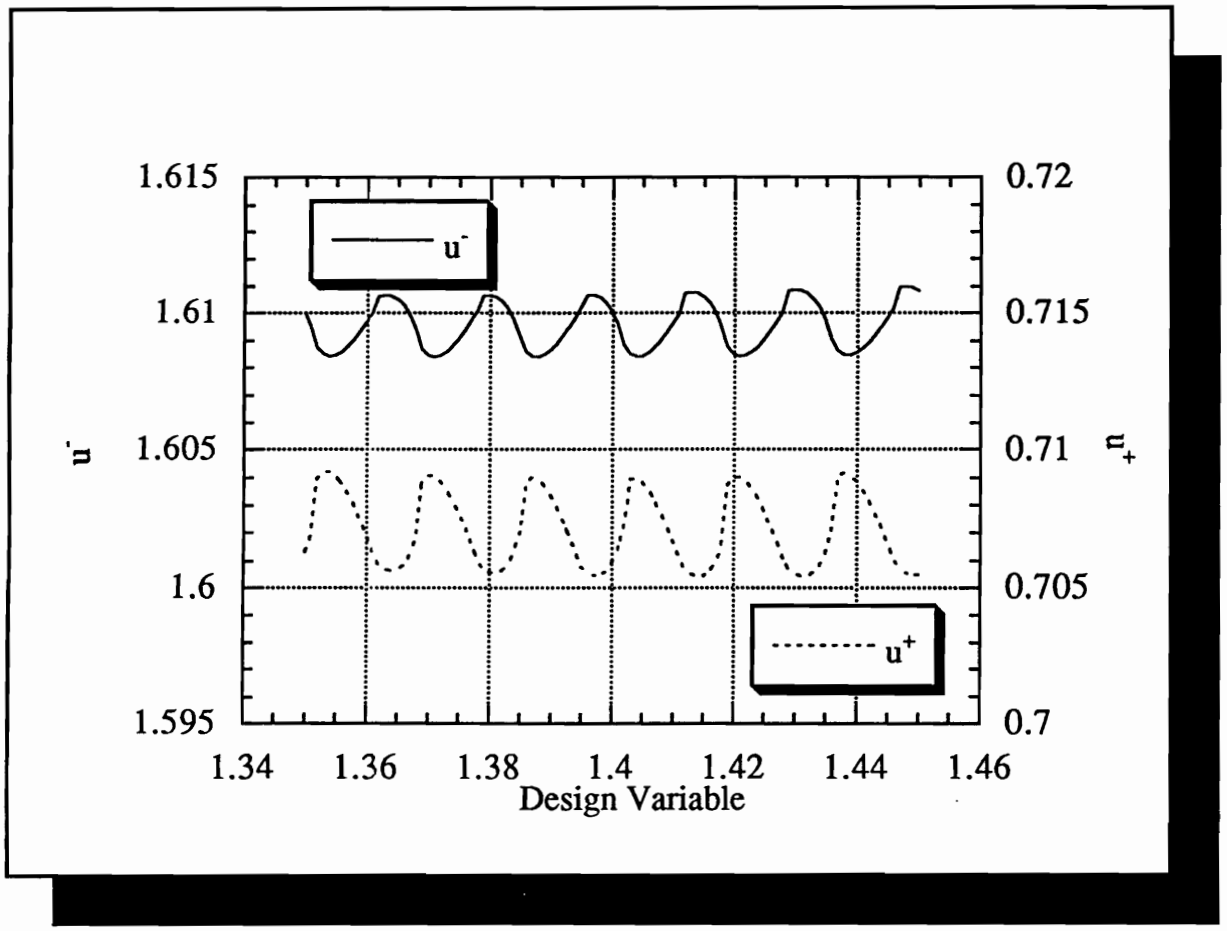

Figure 5.7: Left and right velocity variation at the shock position with design variable (univariate case).

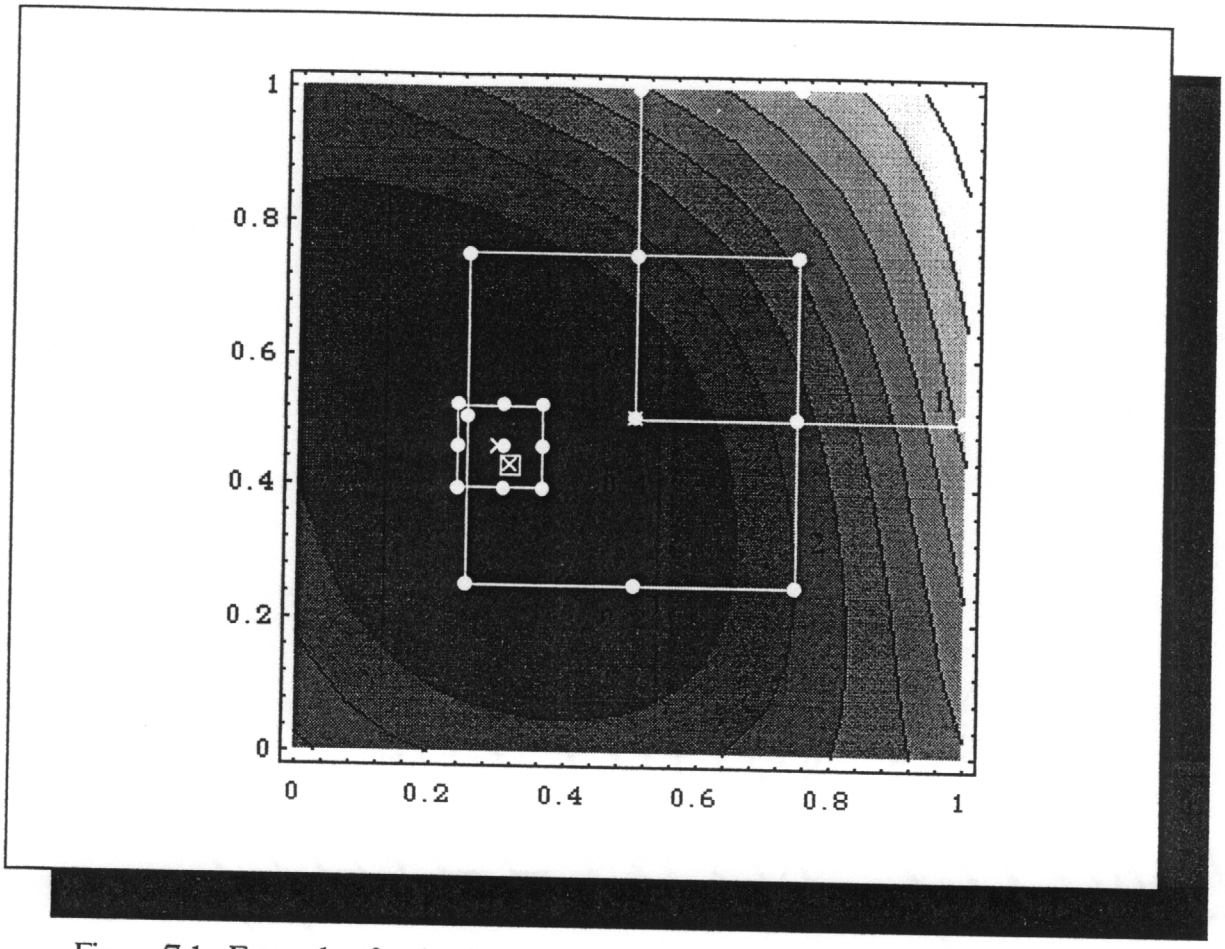

Figure 7.1: Example of reduction and translation of the response surface region during the optimization cycles.

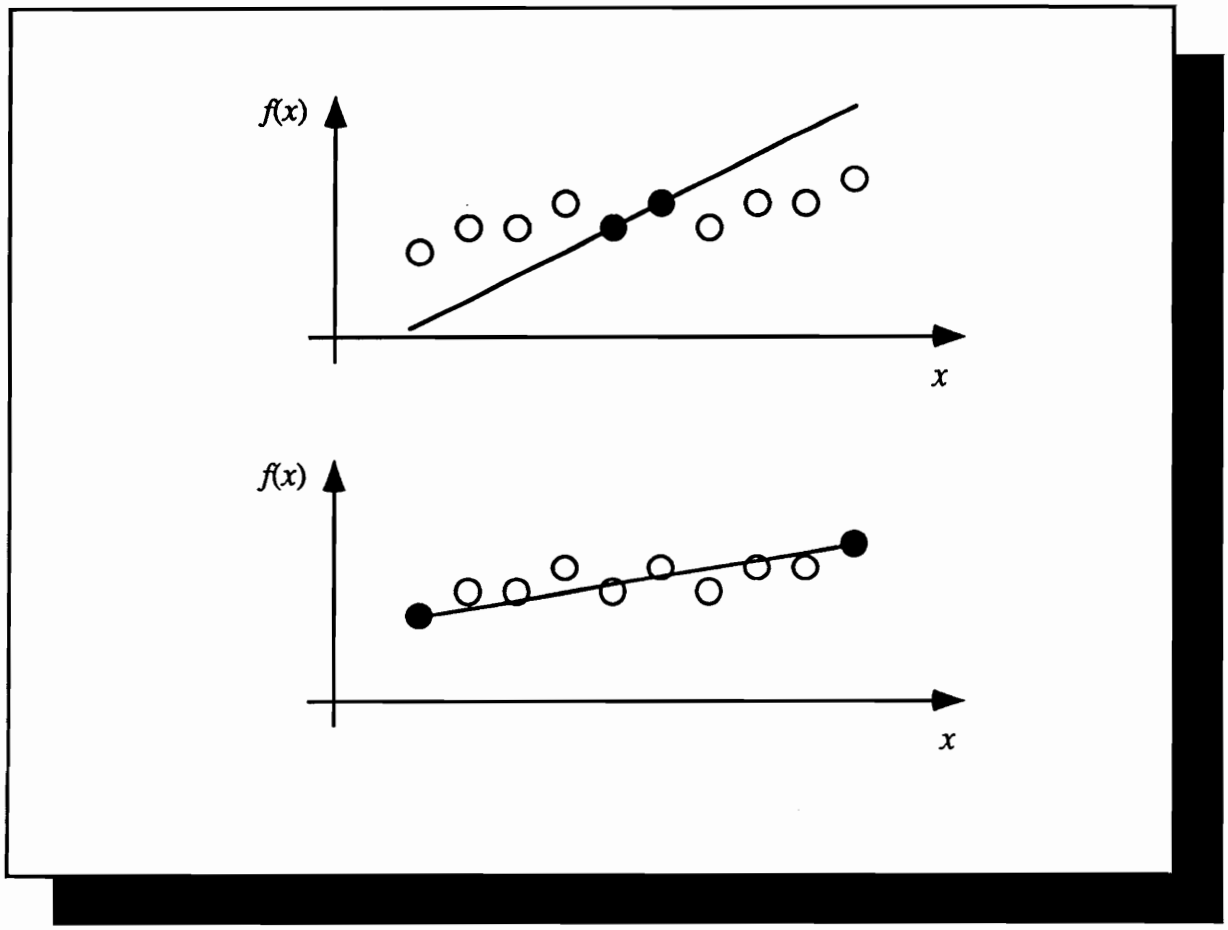

Figure 8.1: Simple example demonstrating how point selection can effect the fidelity of a response surface.

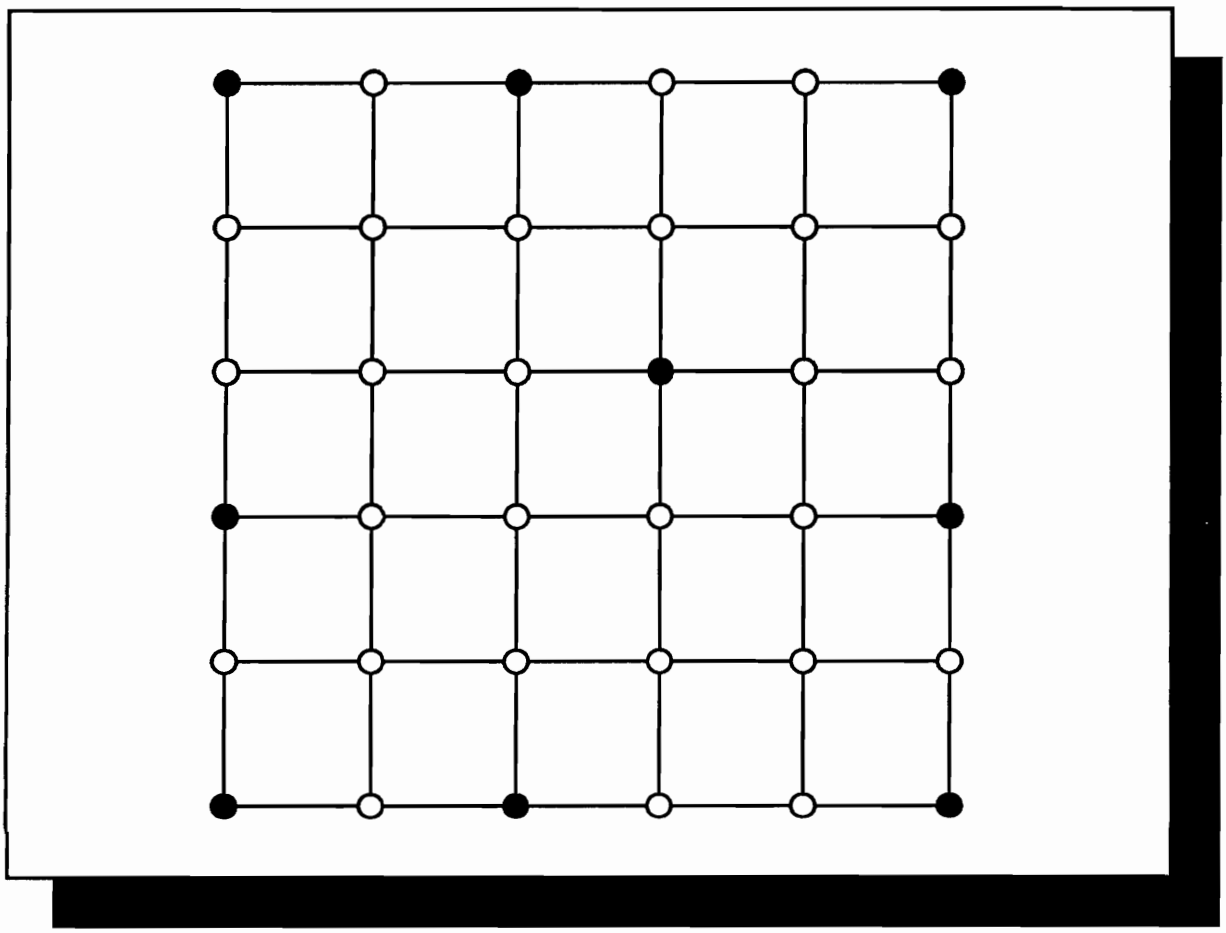

Figure 8.2: D-optimal set of points for $P = c_0 + c_1\xi_1 + c_2\xi_2 + c_3\xi_1^2 + c_4\xi_1\xi_2 + c_5\xi_2^2$ in a rectangular domain.

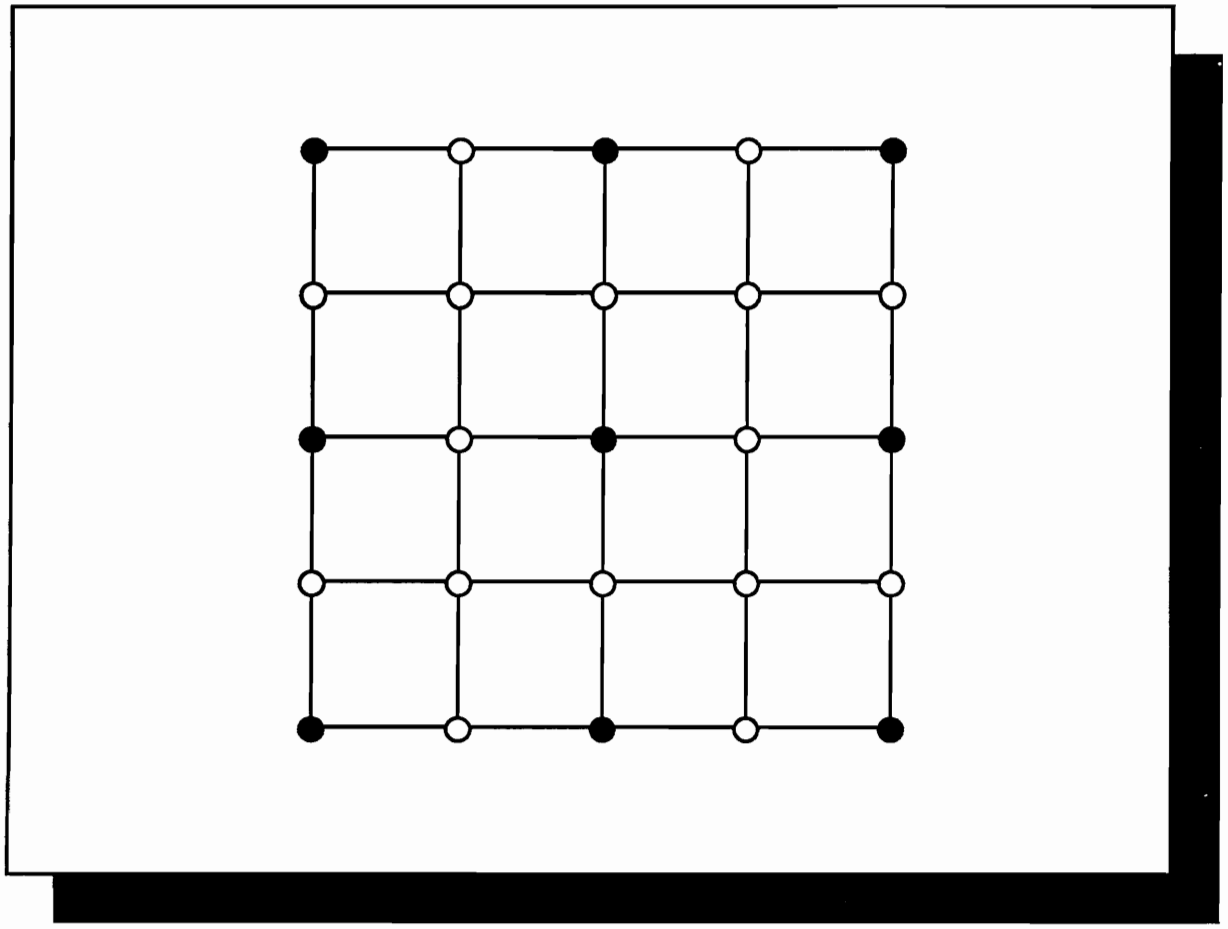

Figure 8.3: D-optimal set of points for a quadratic tensor product in two-dimensional rectangular domain

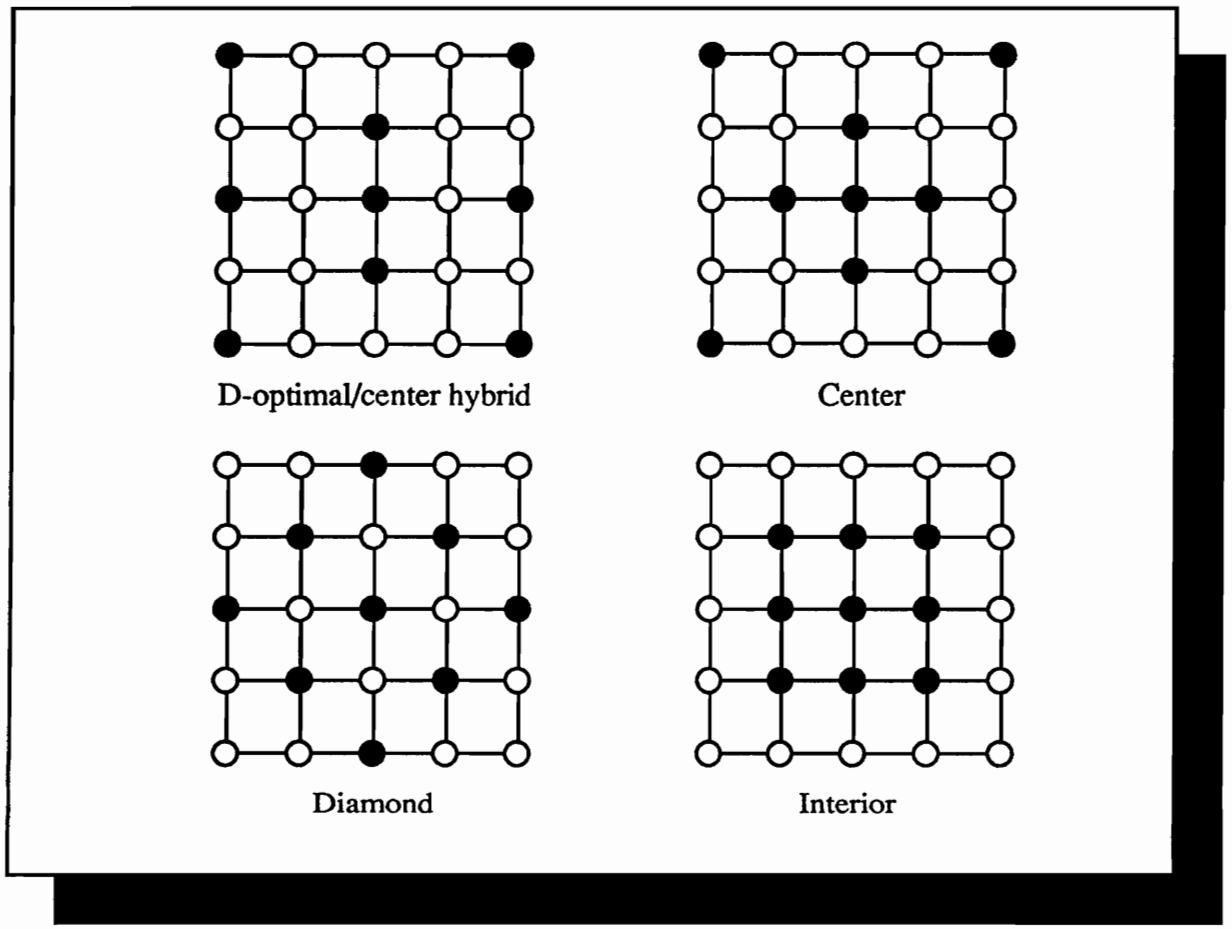

Figure 8.4: Possible designs for constructing a response surface in 2-D rectangular space.

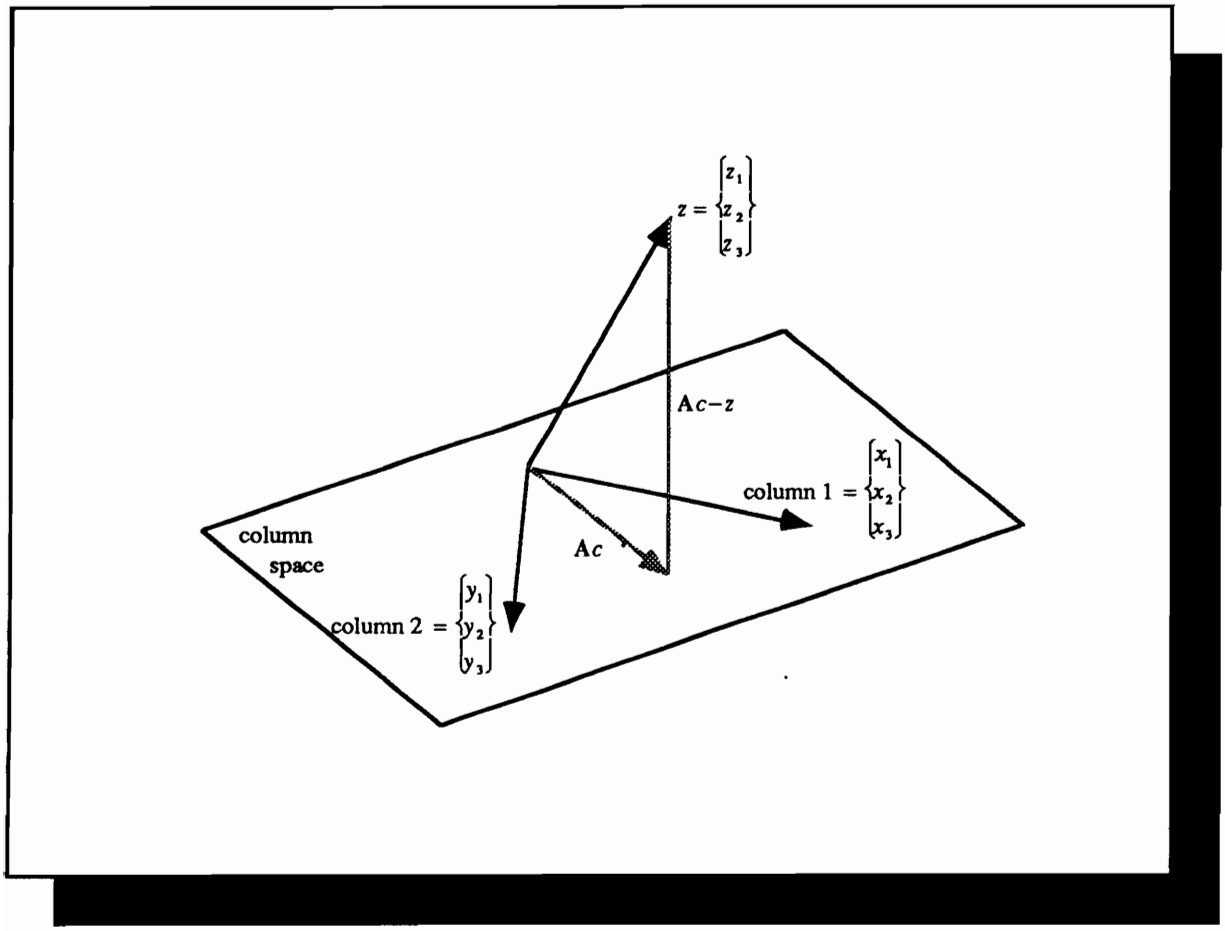

Figure 8.5: Least squares representation in two dimensions shows that the error vector Ac-z is perpendicular to the column space.

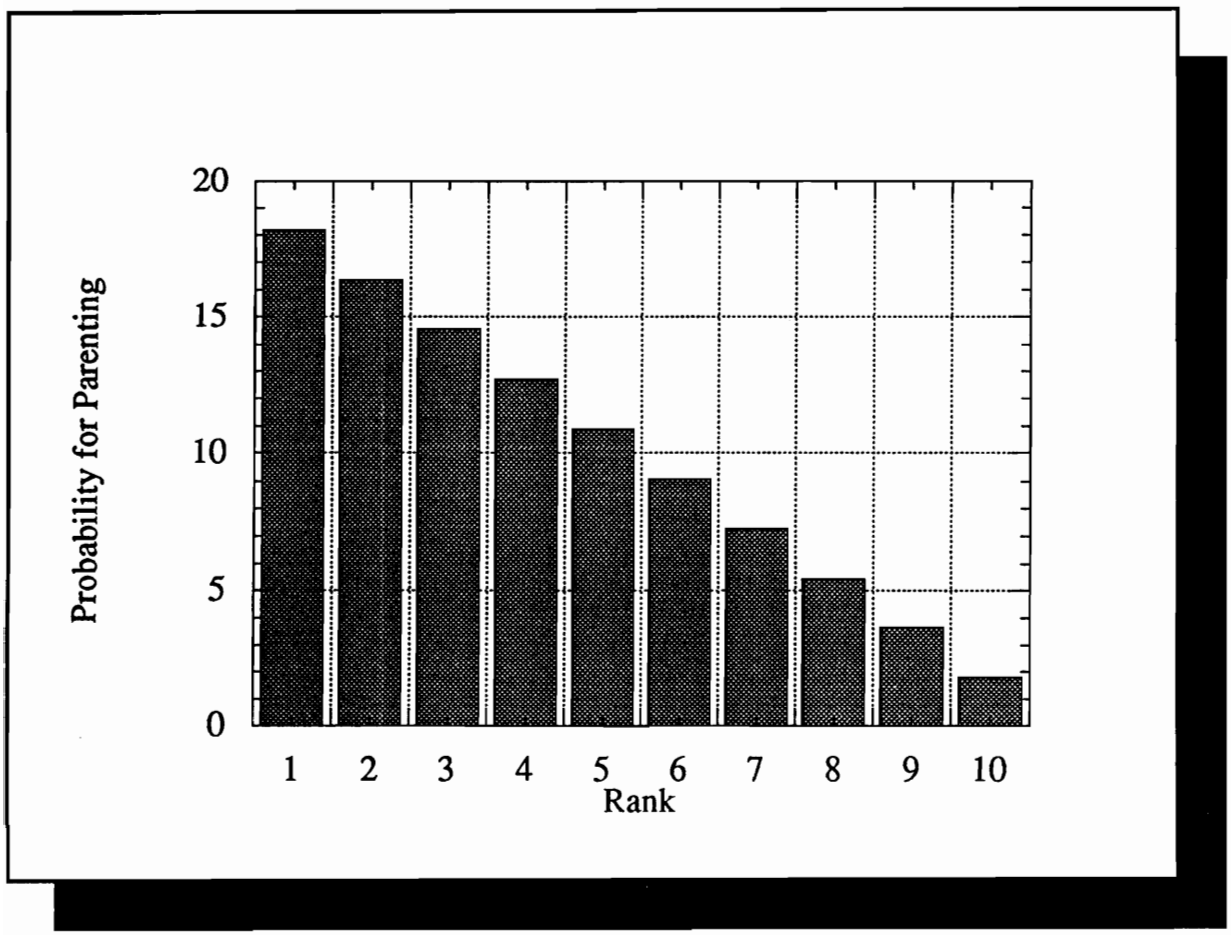

Figure 9.1: Probability of designs being selected for parenting based on the rank in a population of 10 designs.

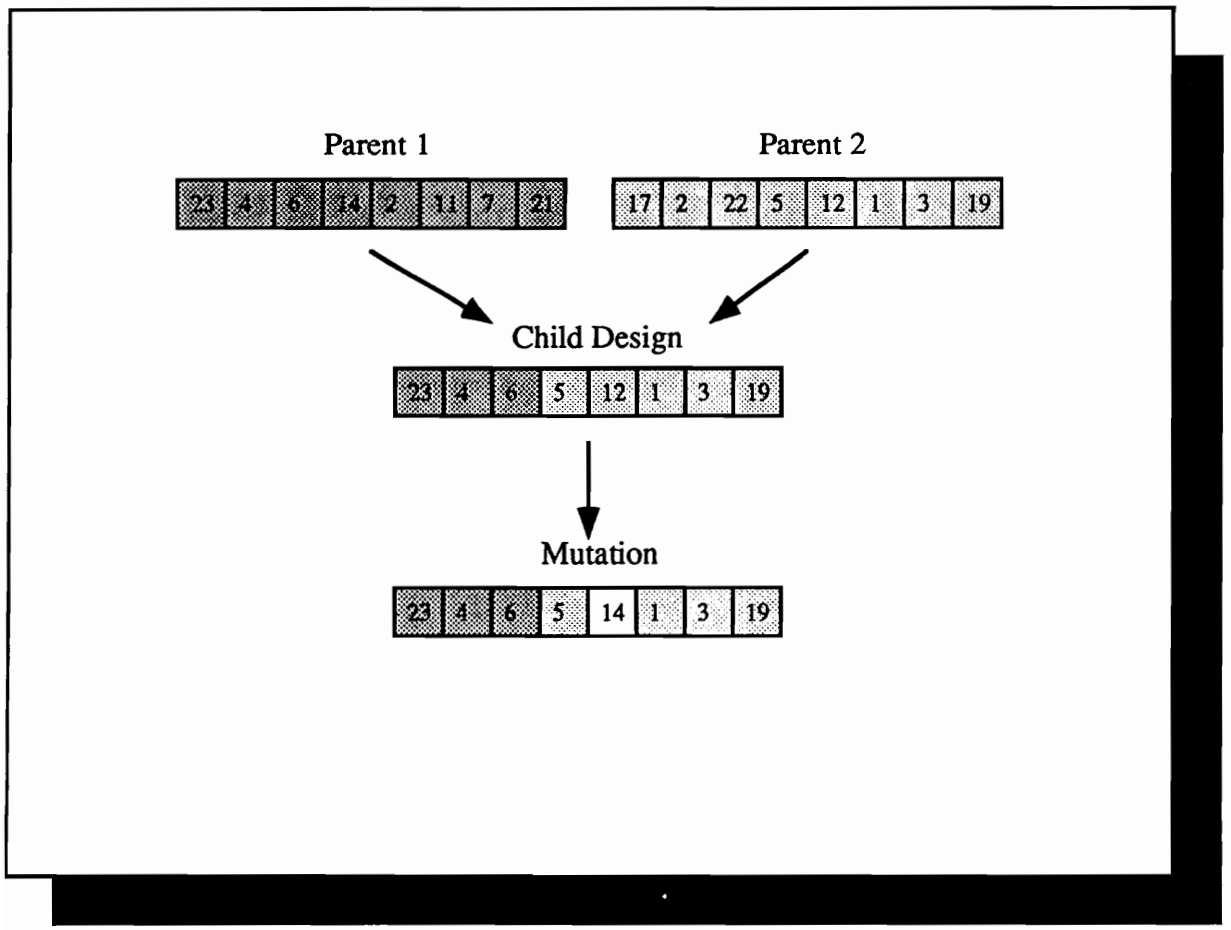

Figure 9.2: Breeding of two parent designs to get one child design.

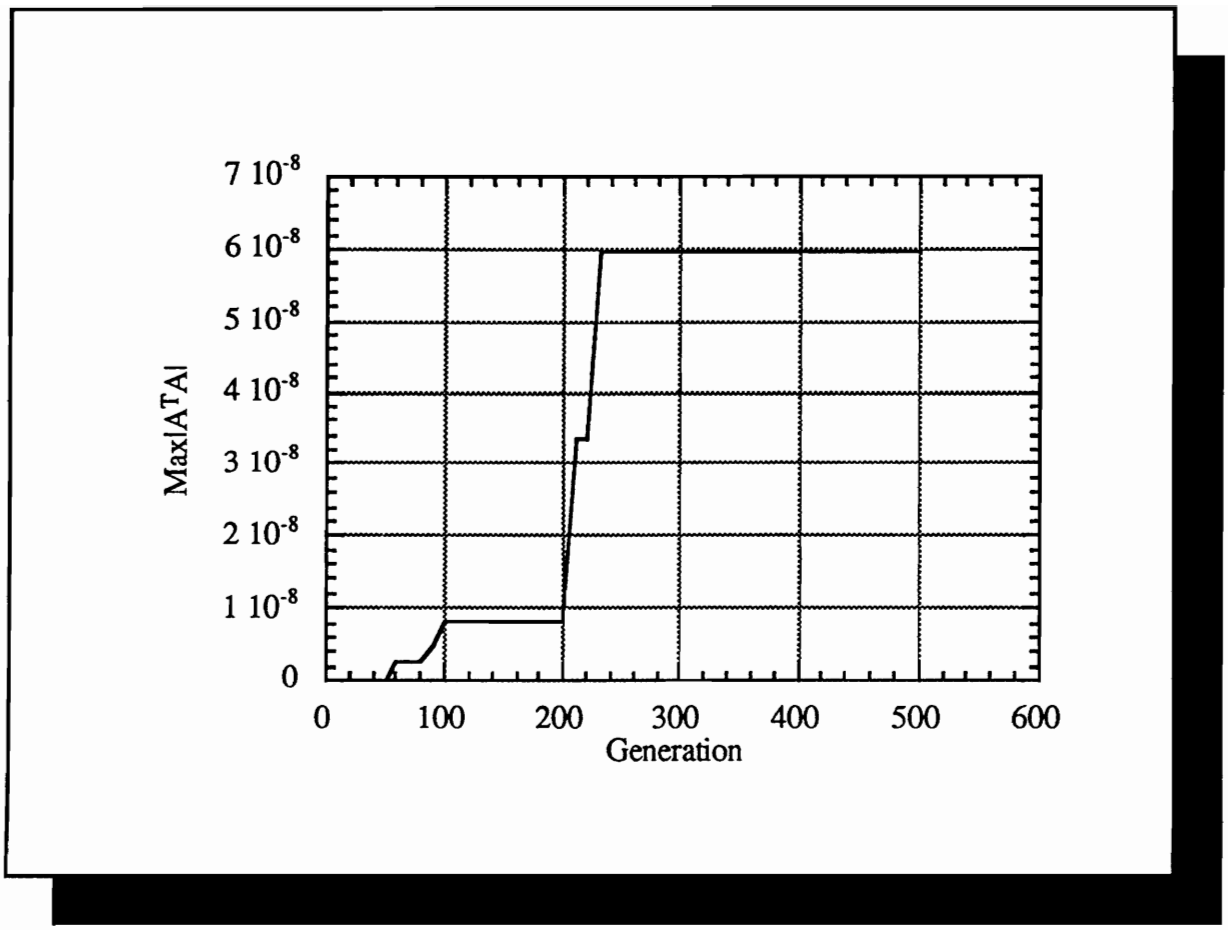

Figure 9.3: GA history of convergence to *D*-optimal set of points for fitting equation 8.3 in square domain

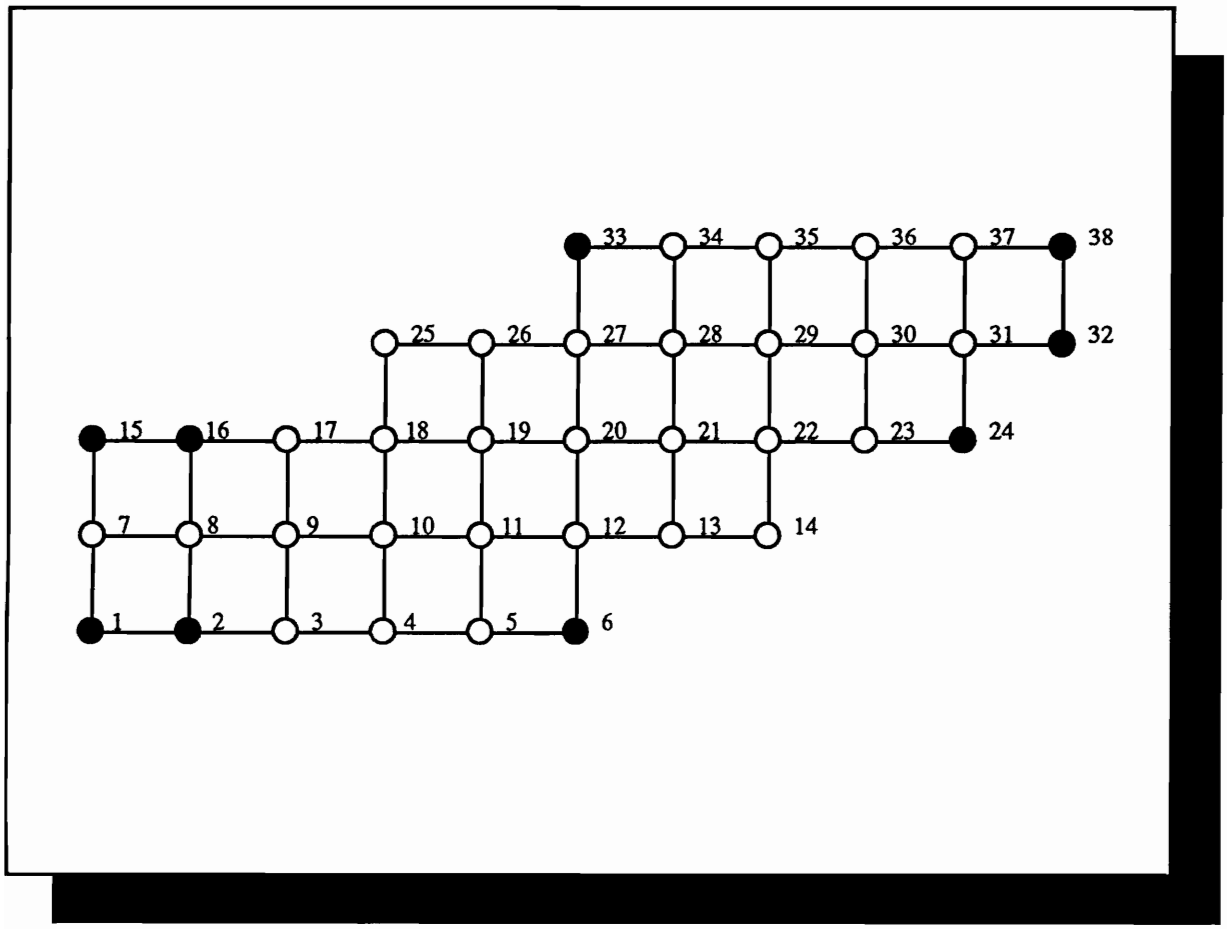

Figure 9.4: *D*-optimal set of points for fitting equation 9.6 in a general shaped domain.

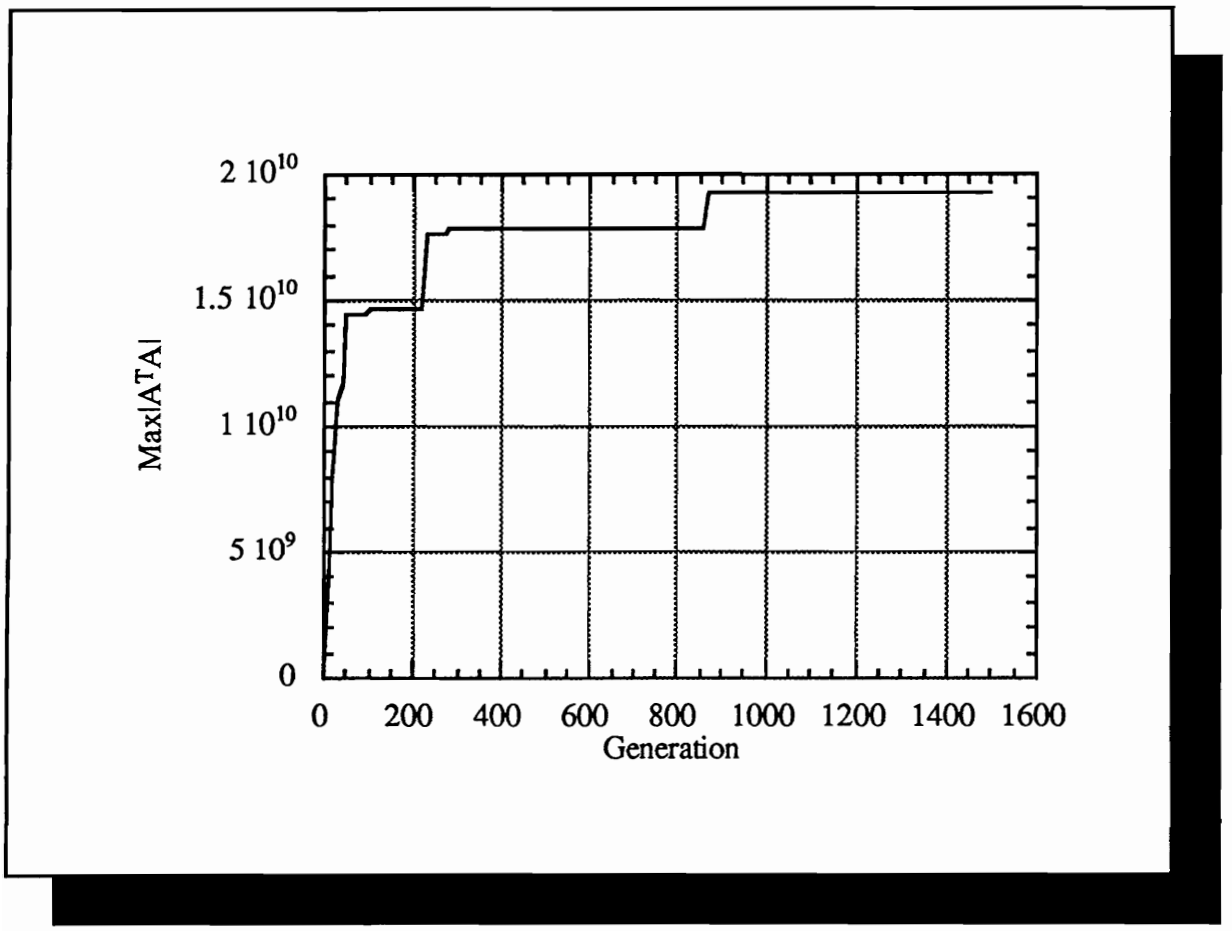

Figure 9.5: GA history of convergence to *D*-optimal set of points for fitting equation 9.6 in the general domain of figure 9.4.

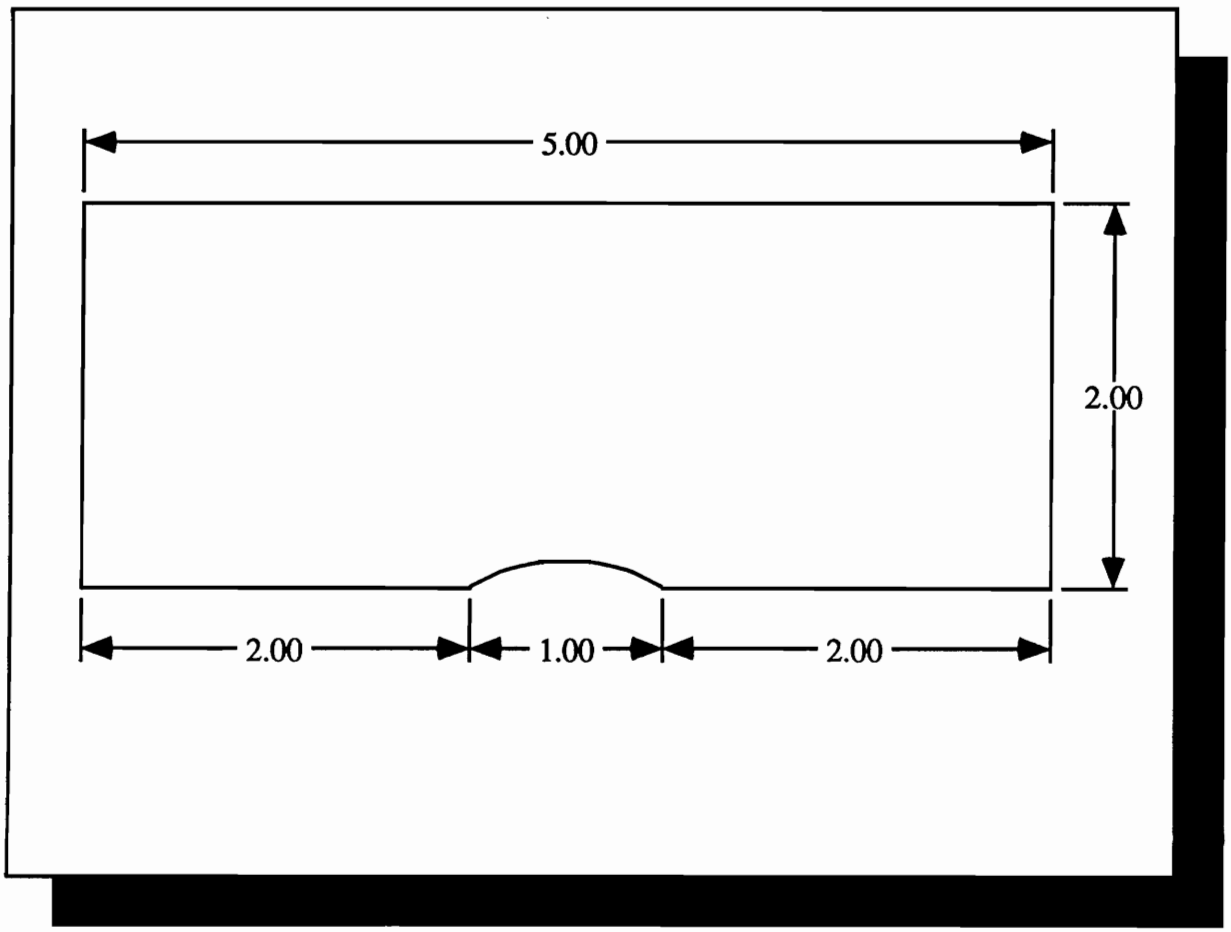

Figure 10.1: Channel geometry.

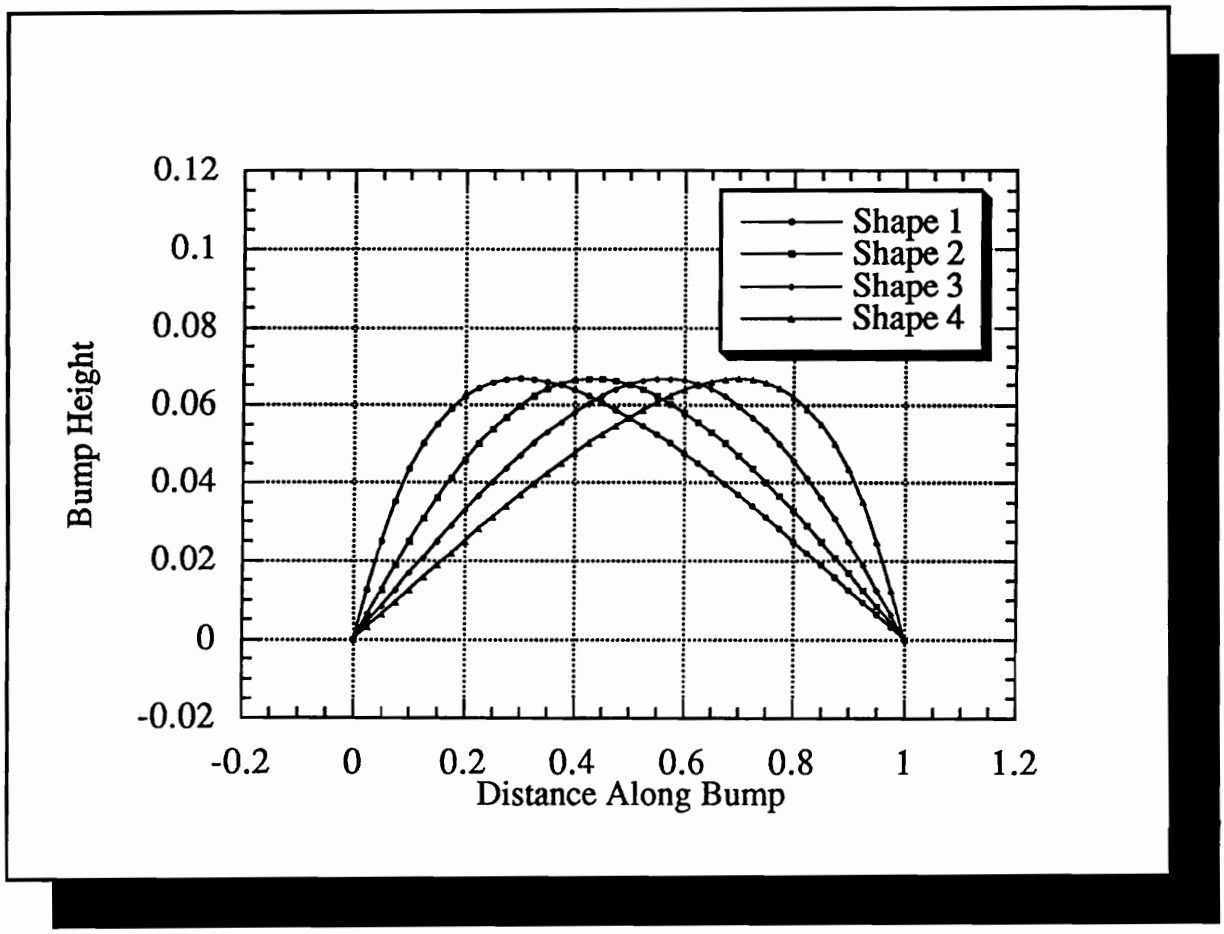

Figure 10.2: Shape functions for the inverse design of a bump in transonic channel flow.

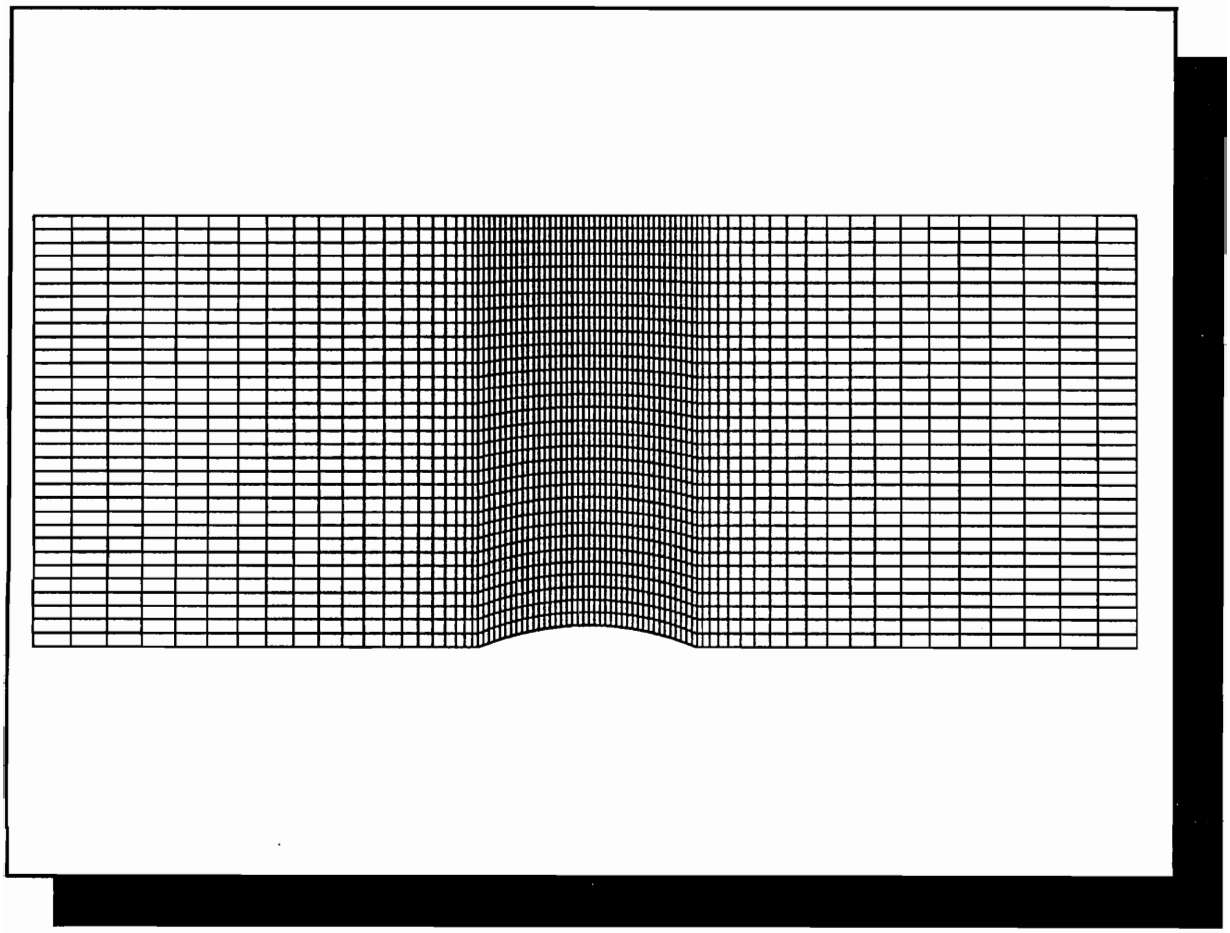

Figure 10.3: Typical grid to generate the Euler solutions for transonic flow through the channel.

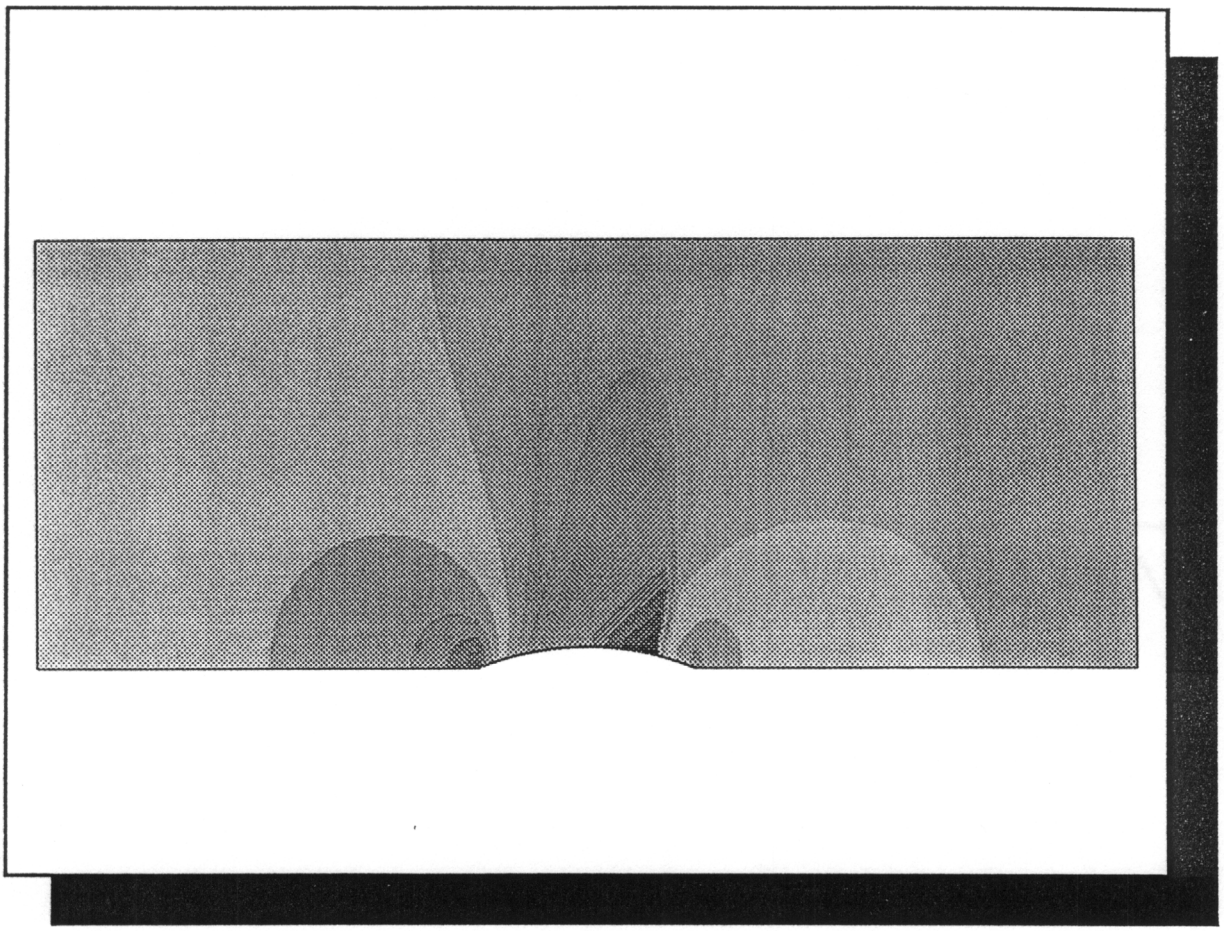

Figure 10.4: Target pressure contours for Mach 0.8 flow through the channel.

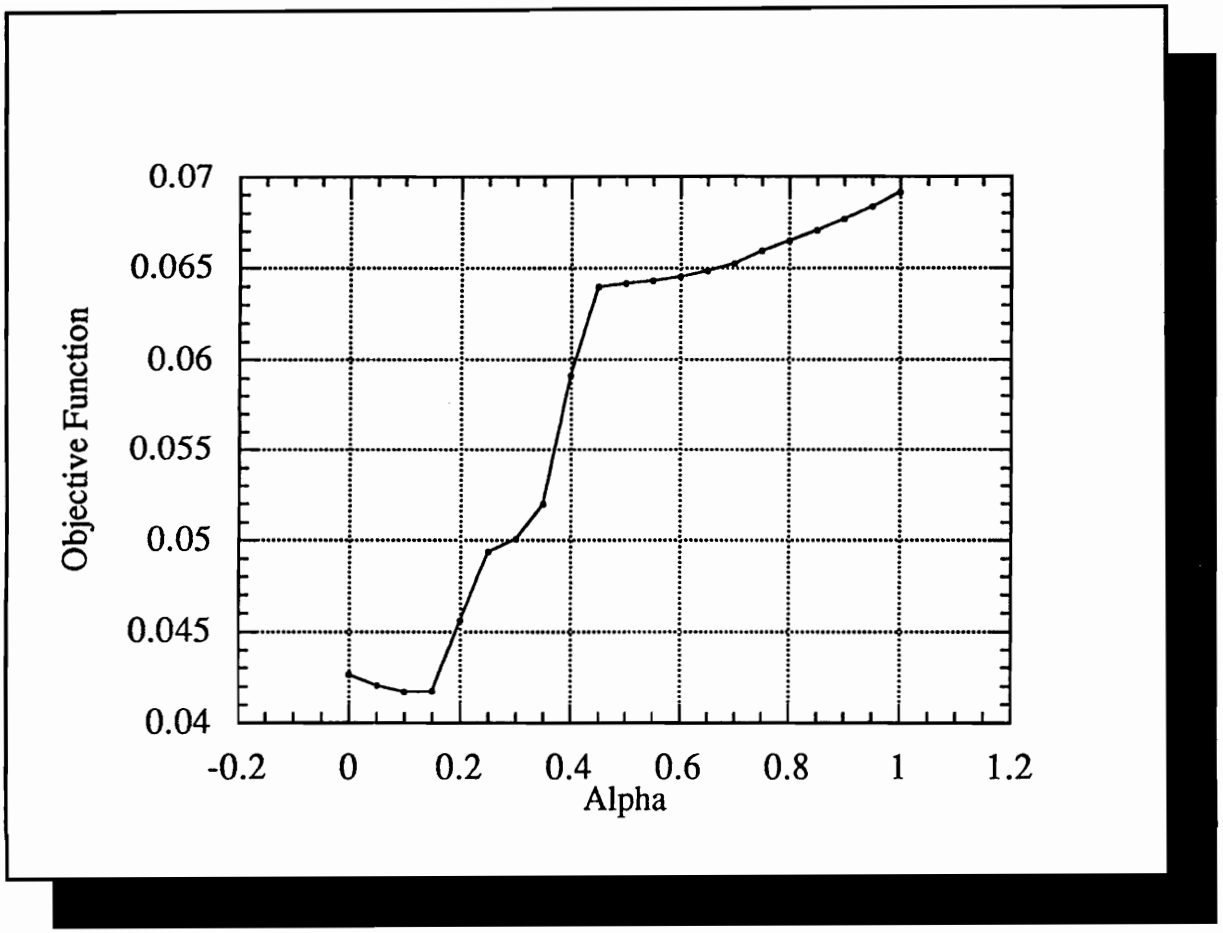

Figure 10.5: One-dimensional cut through the design space of a bump in a channel of transonic flow.

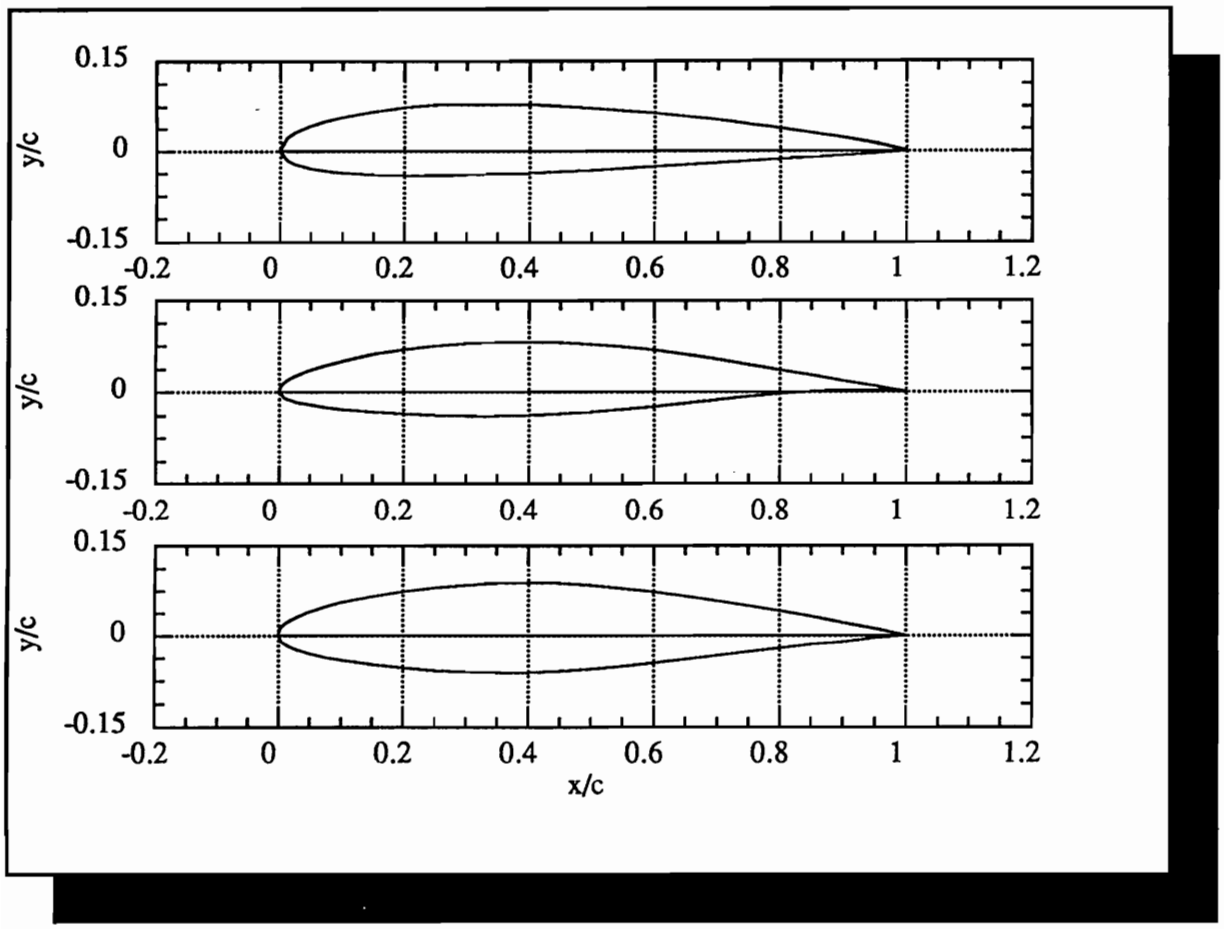

Figure 10.6: 3 of 6 shape functions for the transonic airfoil design problem.

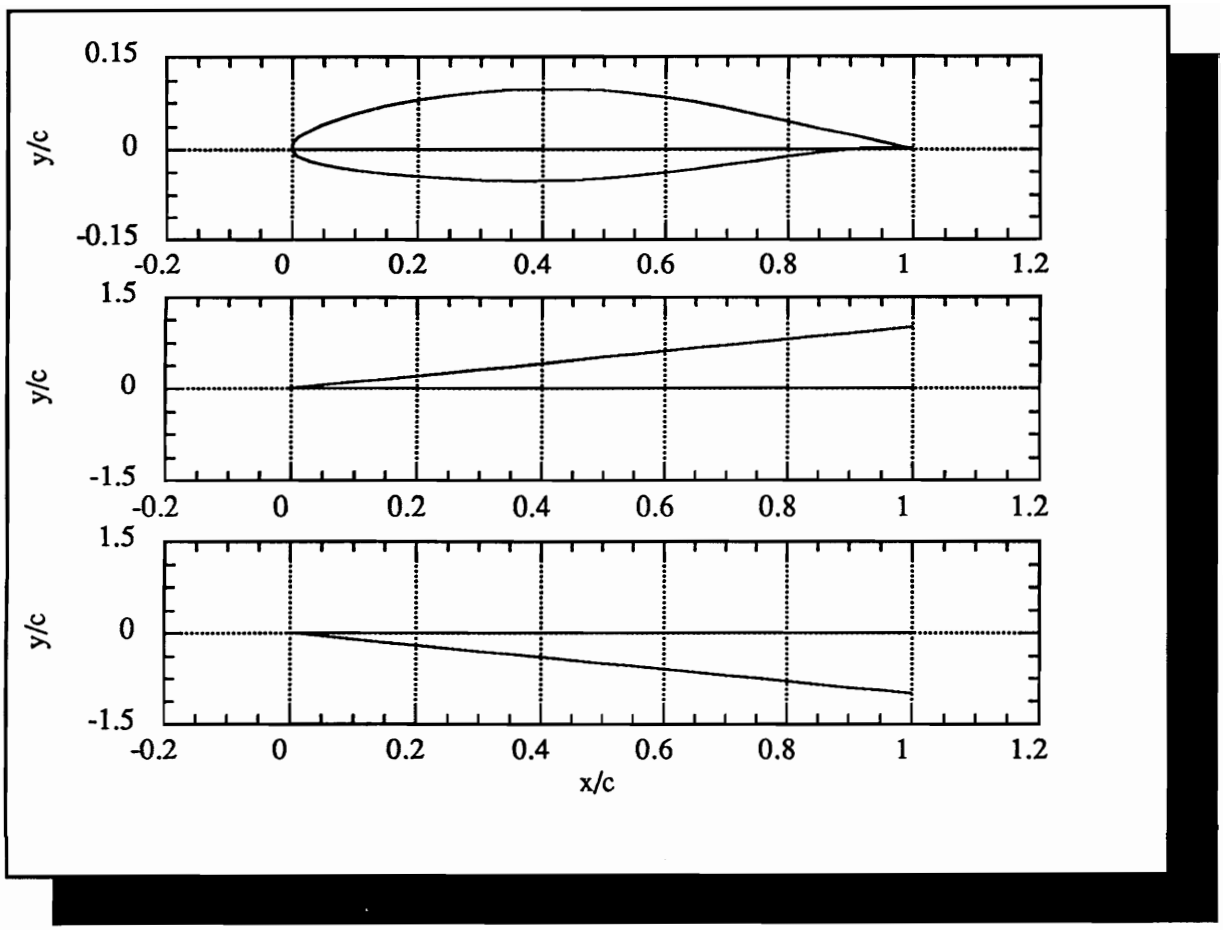

Figure 10.7: 3 of 6 shape functions for the transonic airfoil design problem.

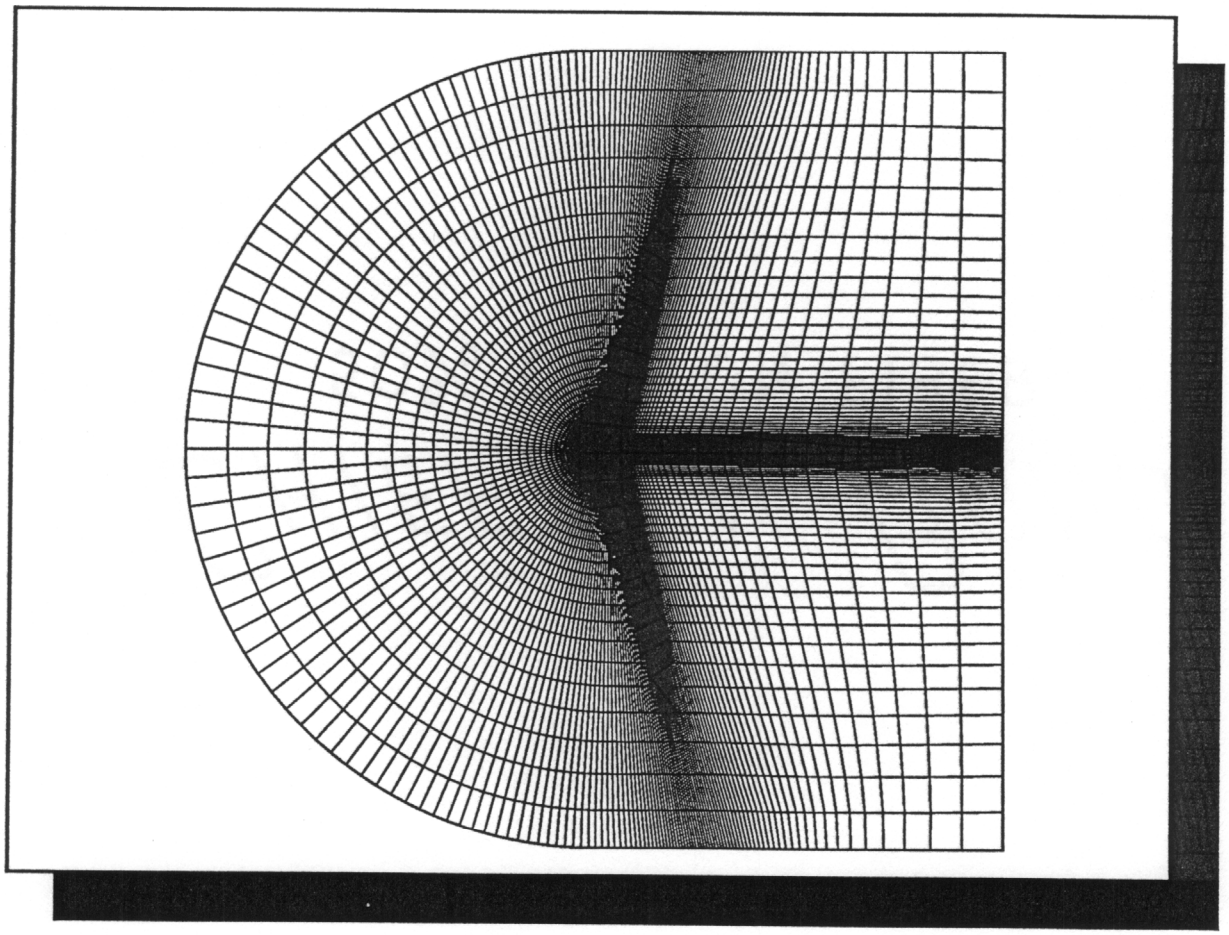

Figure 10.8: Typical grid to generate the Euler solutions for transonic flow over an airfoil.

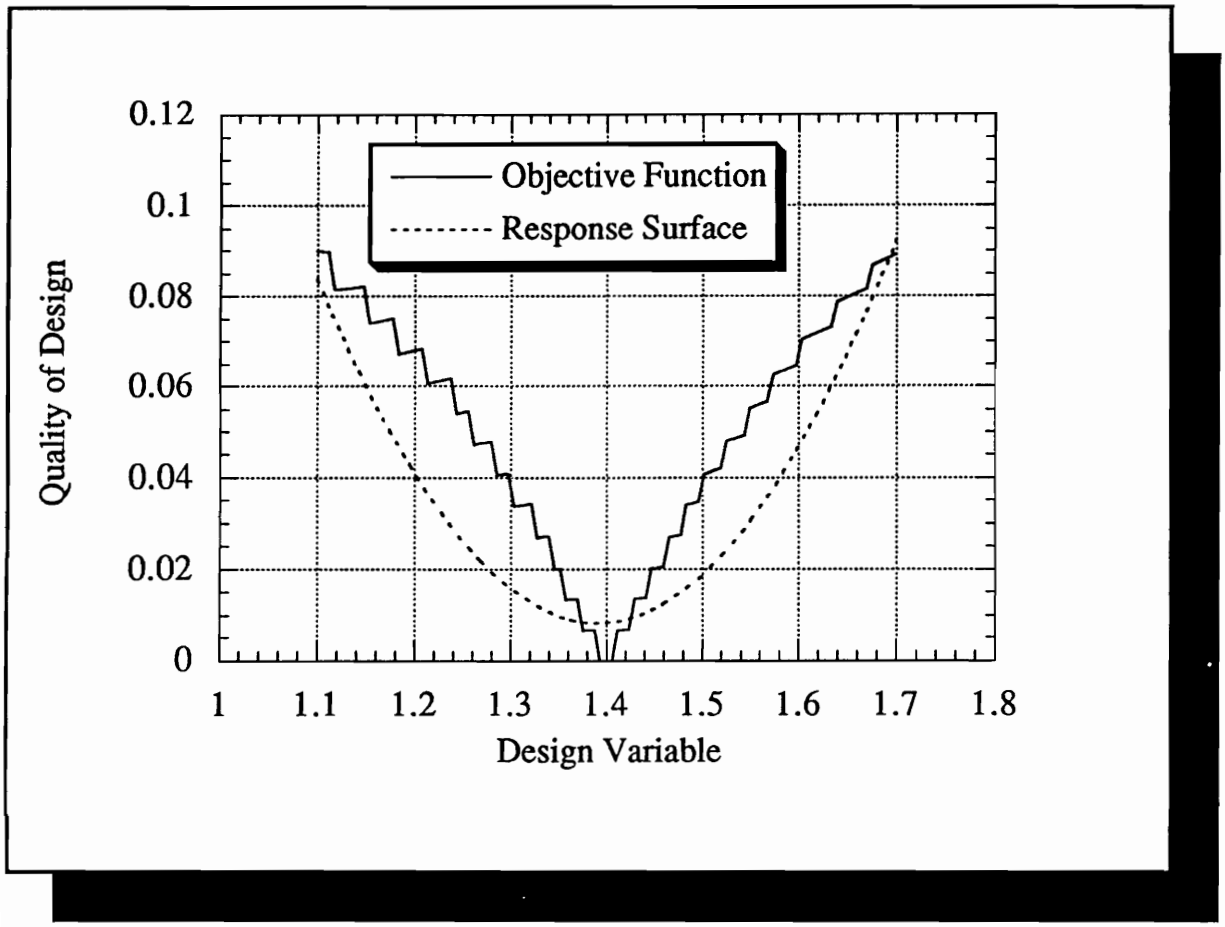

Figure 11.1: Response surface modeling the objective function of the one-dimensional duct problem parameterized by one design variable.

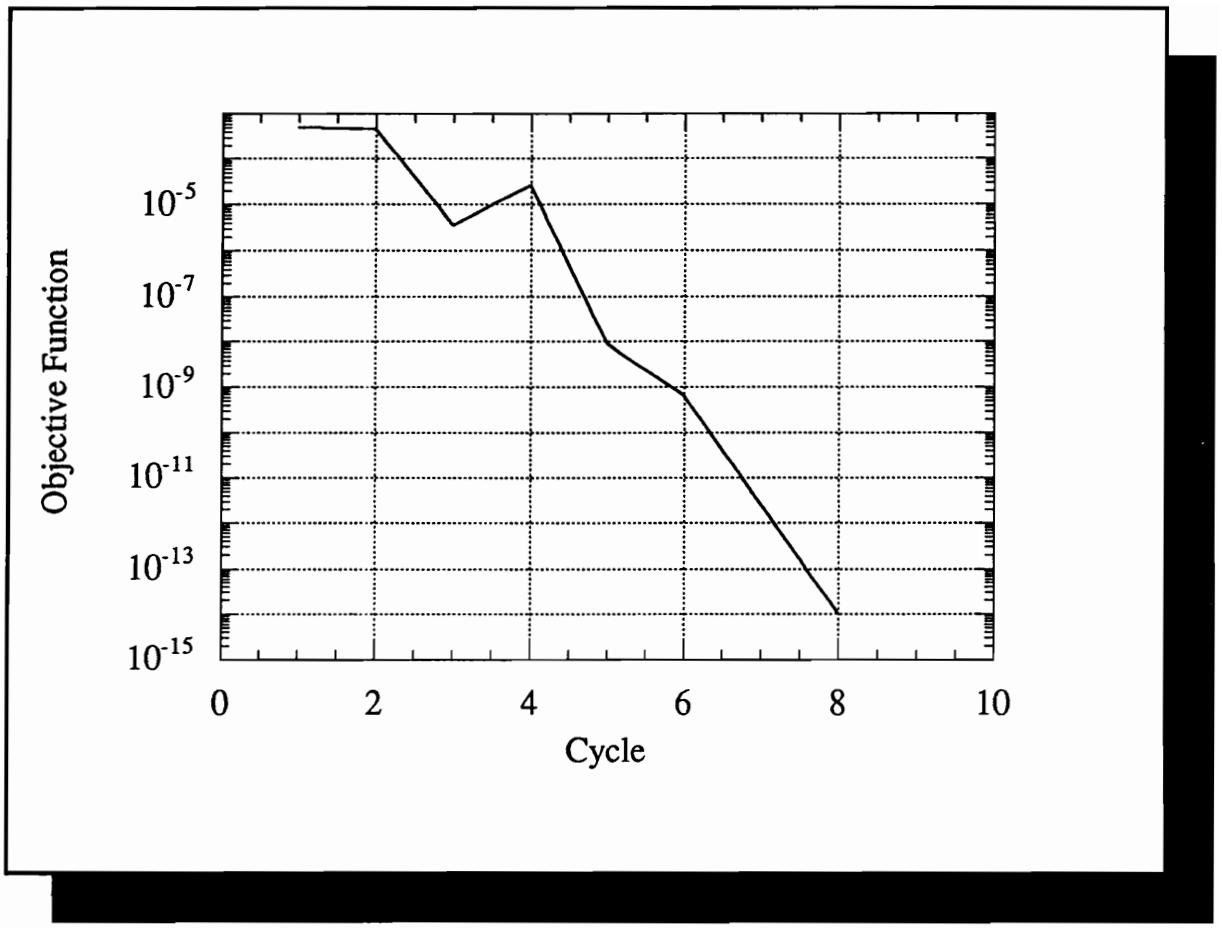

Figure 11.2: Convergence history of the one-dimensional duct problem parameterized by one design variable and optimized by response surfaces.

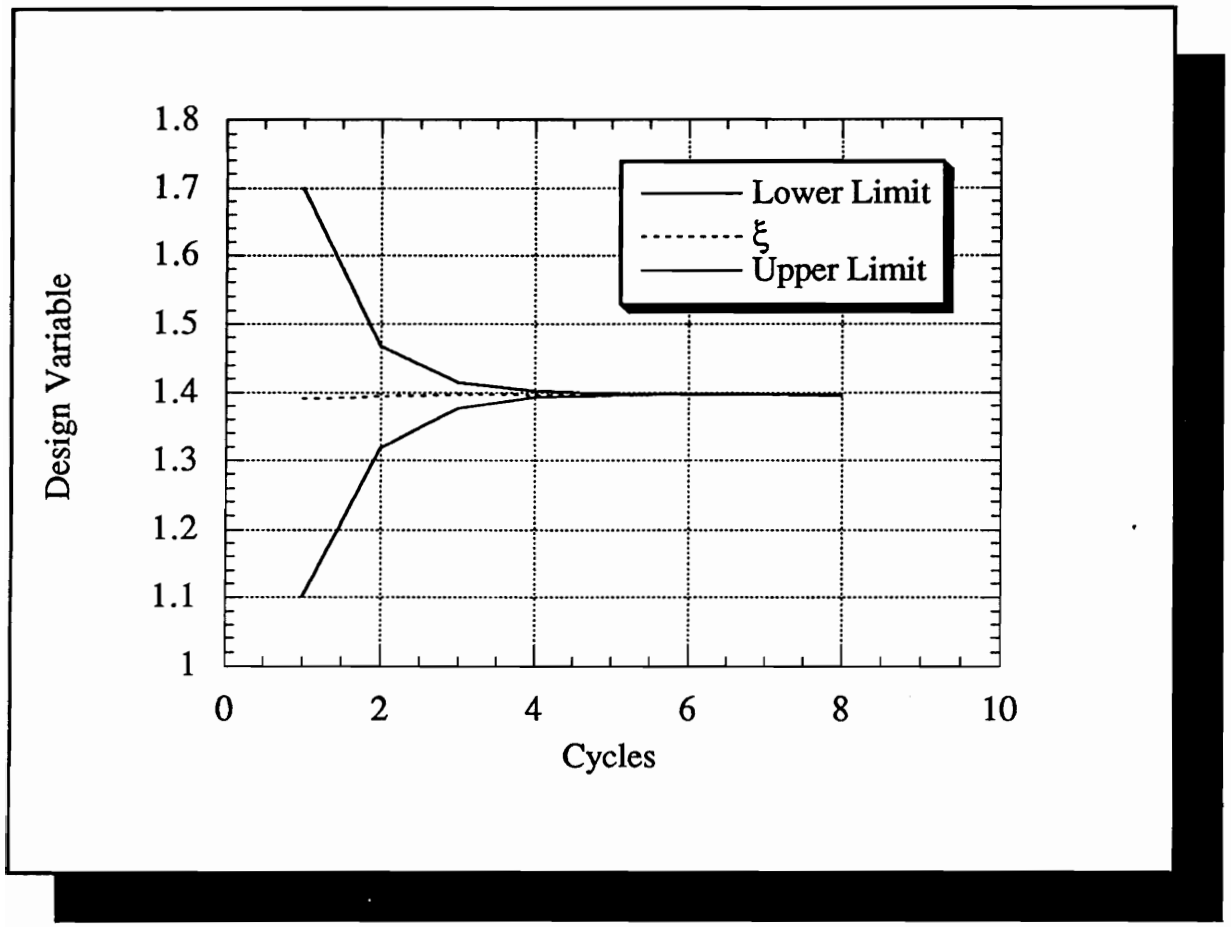

Figure 11.3: Convergence of the design variable for the one-dimensional duct problem optimized by response surfaces.

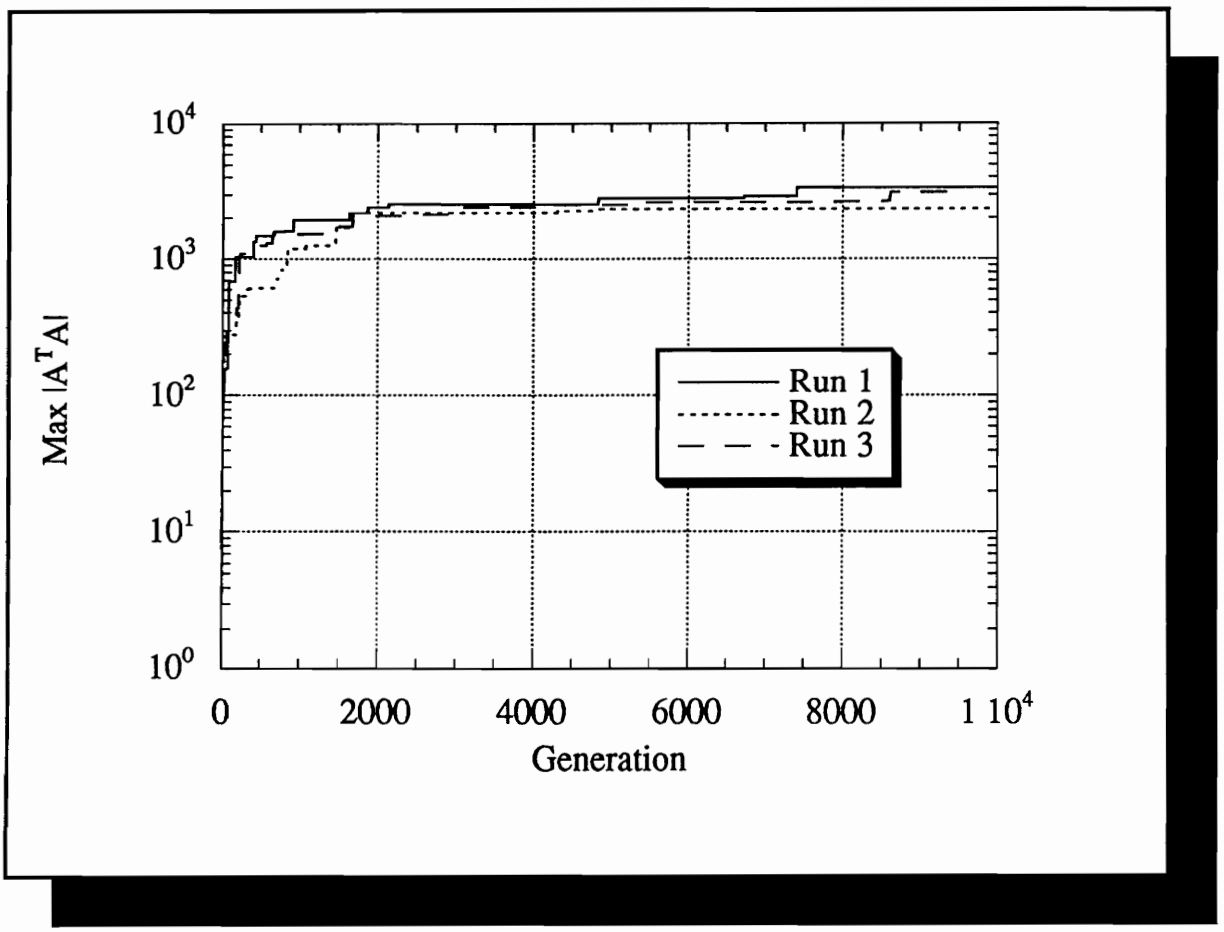

Figure 11.4: GA history of convergence to D-optimal set of points for fitting a quadratic in the region defined by (11.6).

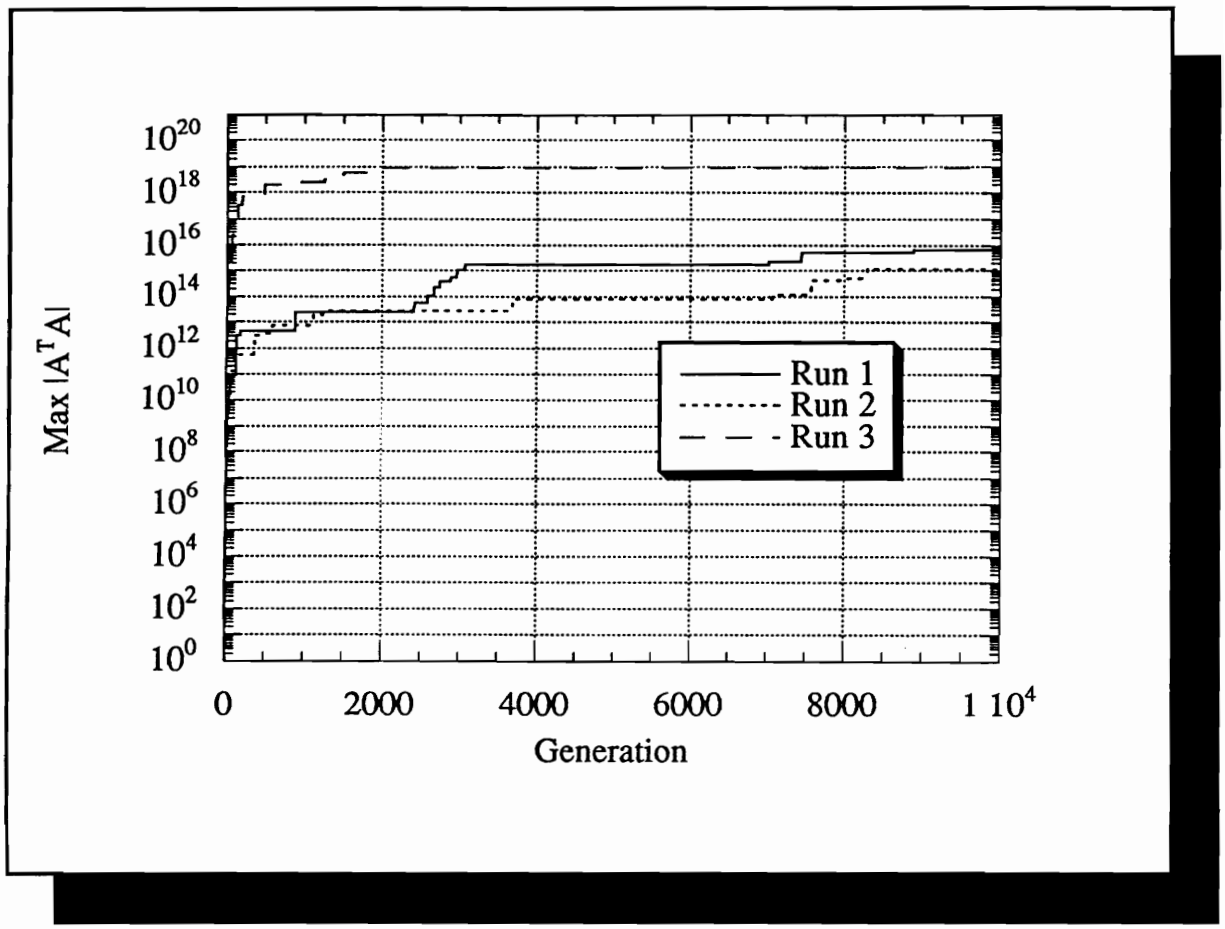

Figure 11.5: GA history of convergence to D-optimal set of points for fitting a quadratic tensor product in the region defined by (11.6).

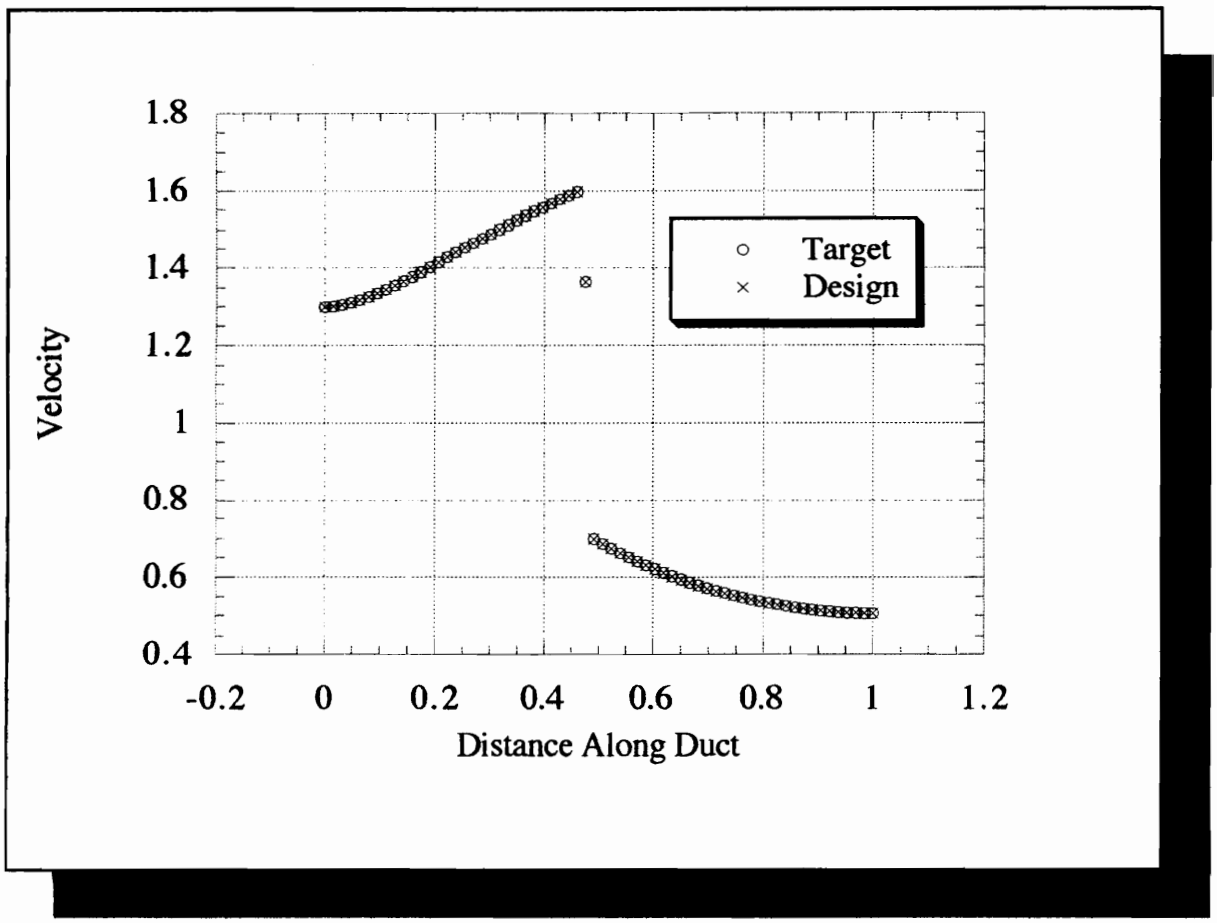

Figure 11.6: Response surface optimization result for the three design variable parameterization of the duct shows the shocks aligned.

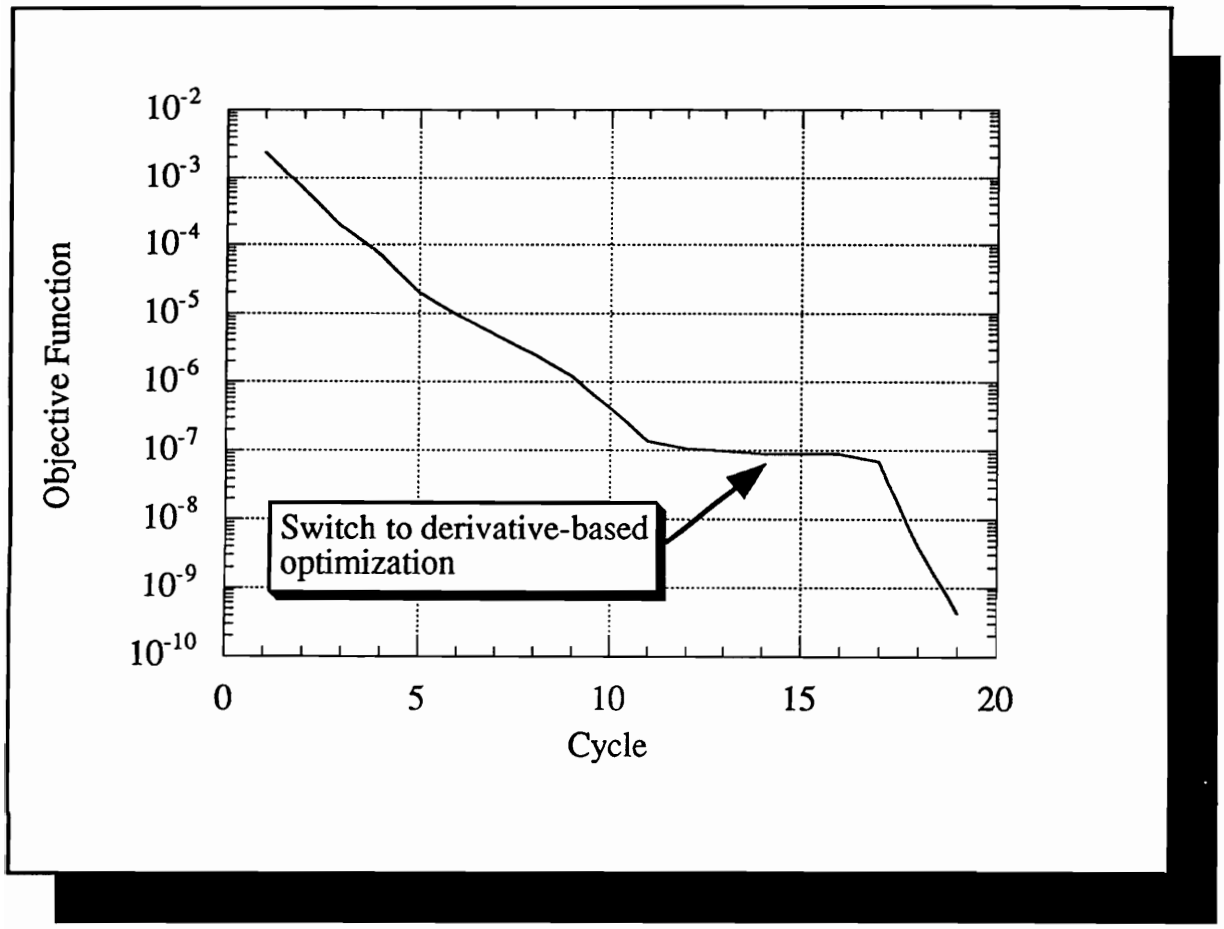

Figure 11.7: Convergence history of the one-dimensional duct problem parameterized by three design variable and optimized by quadratic response surfaces followed by derivative-based optimization.

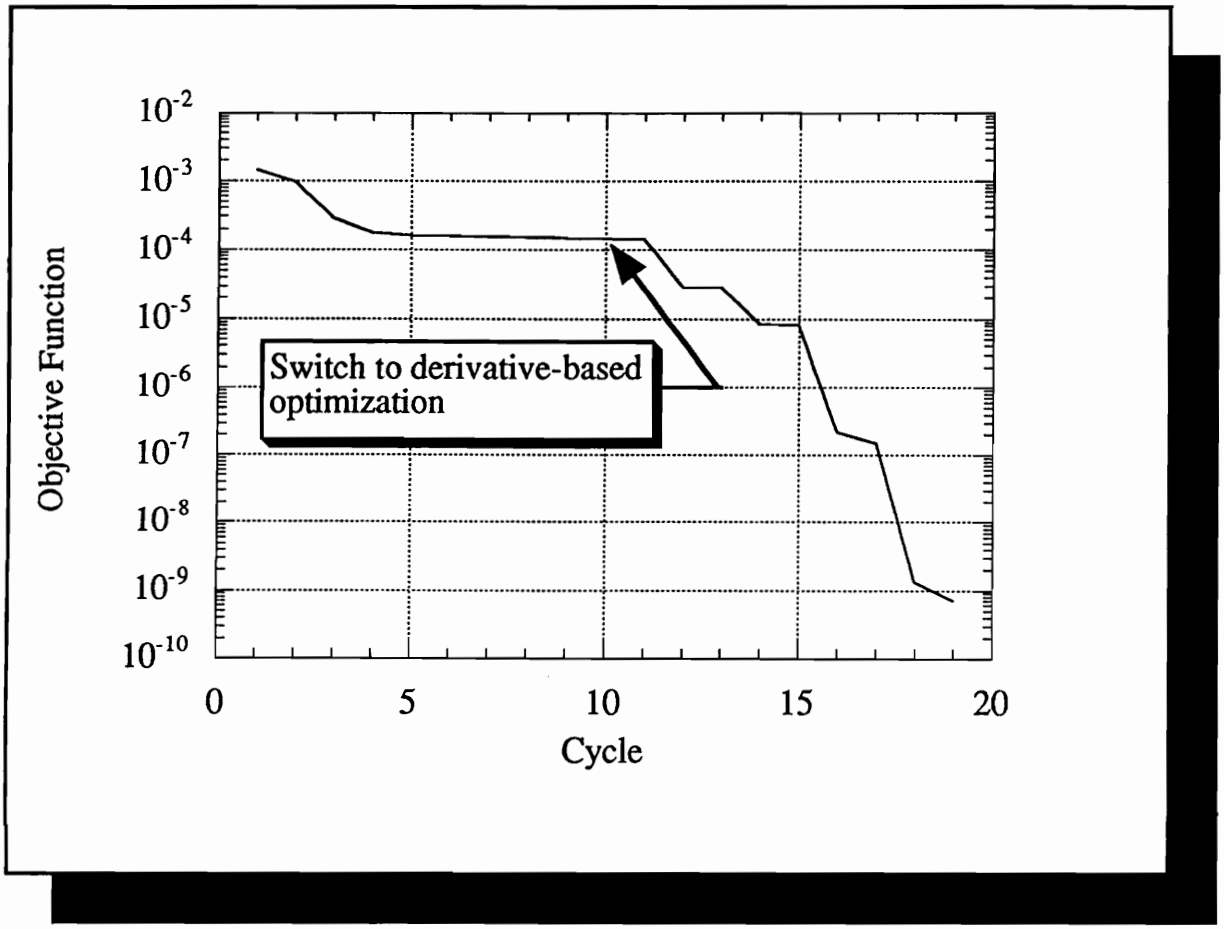

Figure 11.8: Convergence history of the one-dimensional duct problem parameterized by three design variable and optimized by quadratic tensor product response surfaces followed by derivative-based optimization.

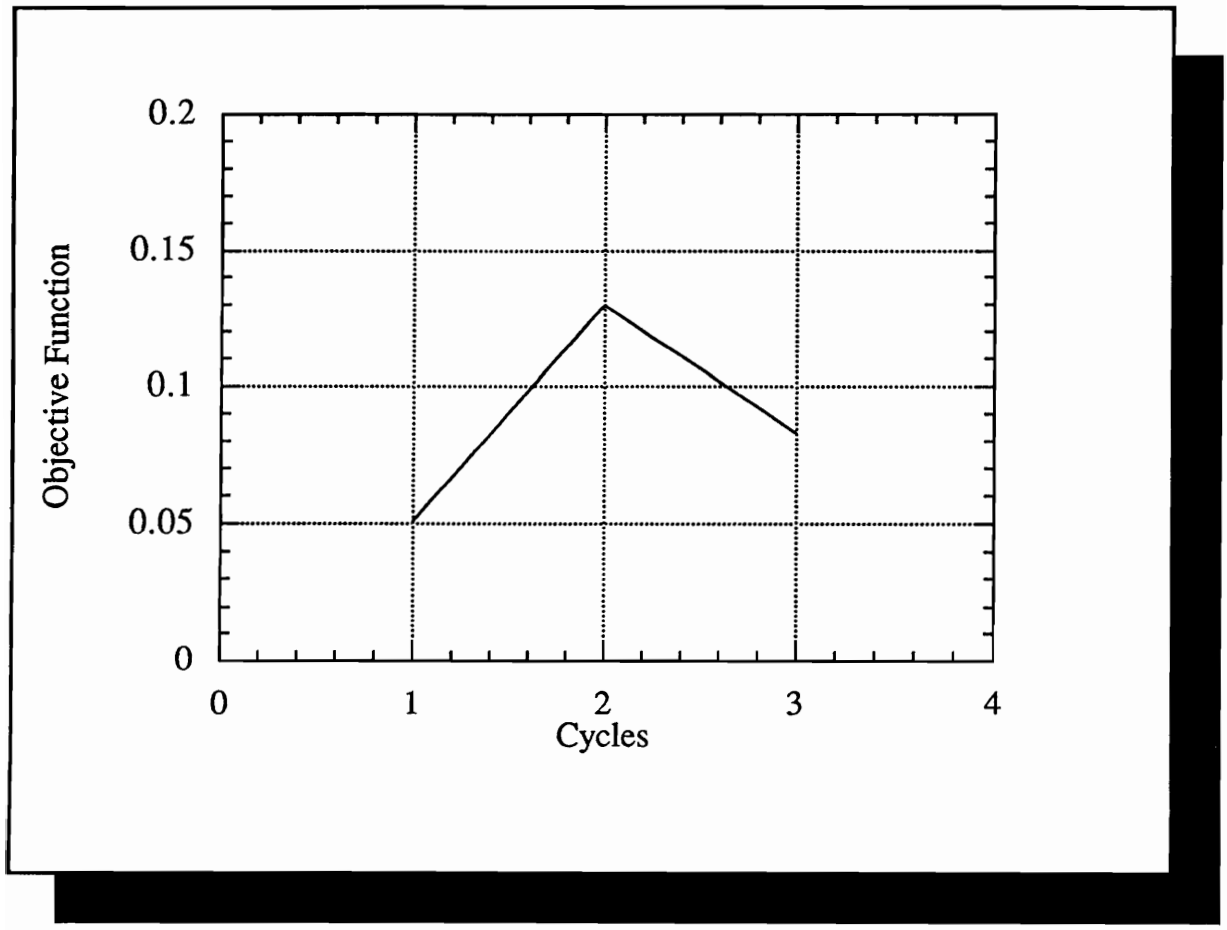

Figure 11.9: Convergence history of the transonic bump problem optimized with response surfaces.

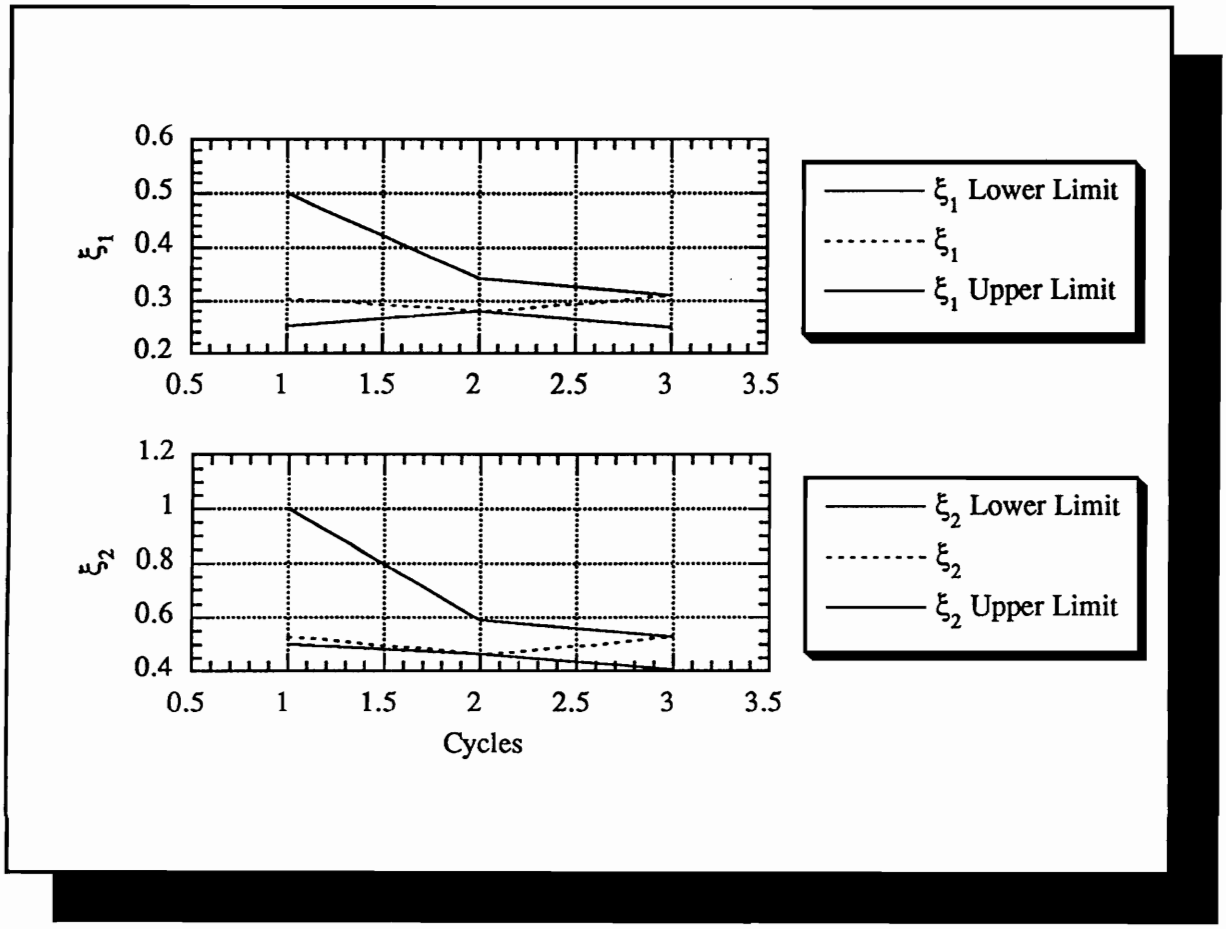

Figure 11.10: Convergence of ξ_1 and ξ_2 for the transonic bump problem optimized by response surfaces.

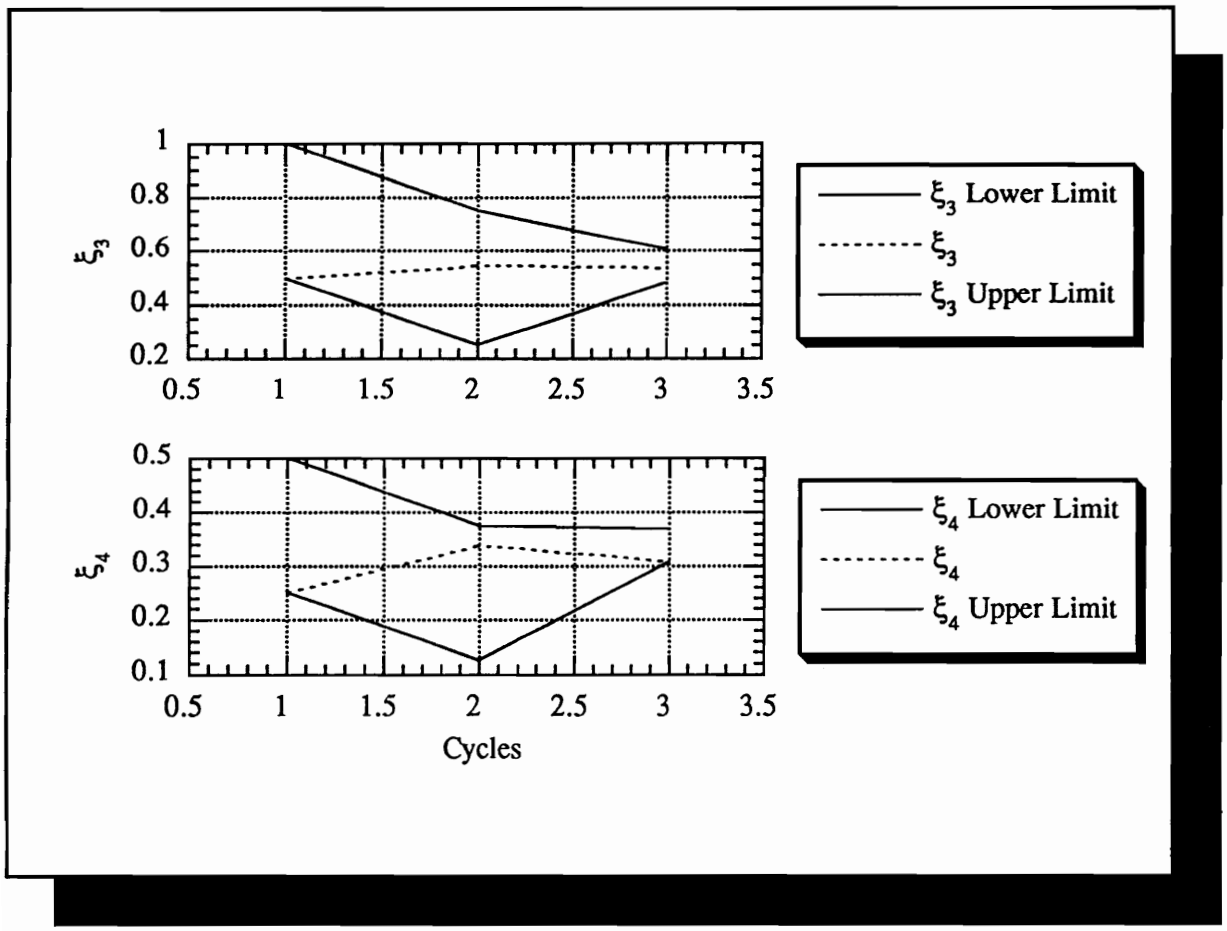

Figure 11.11: Convergence of ξ_3 and ξ_4 for the transonic bump problem optimized by response surfaces.

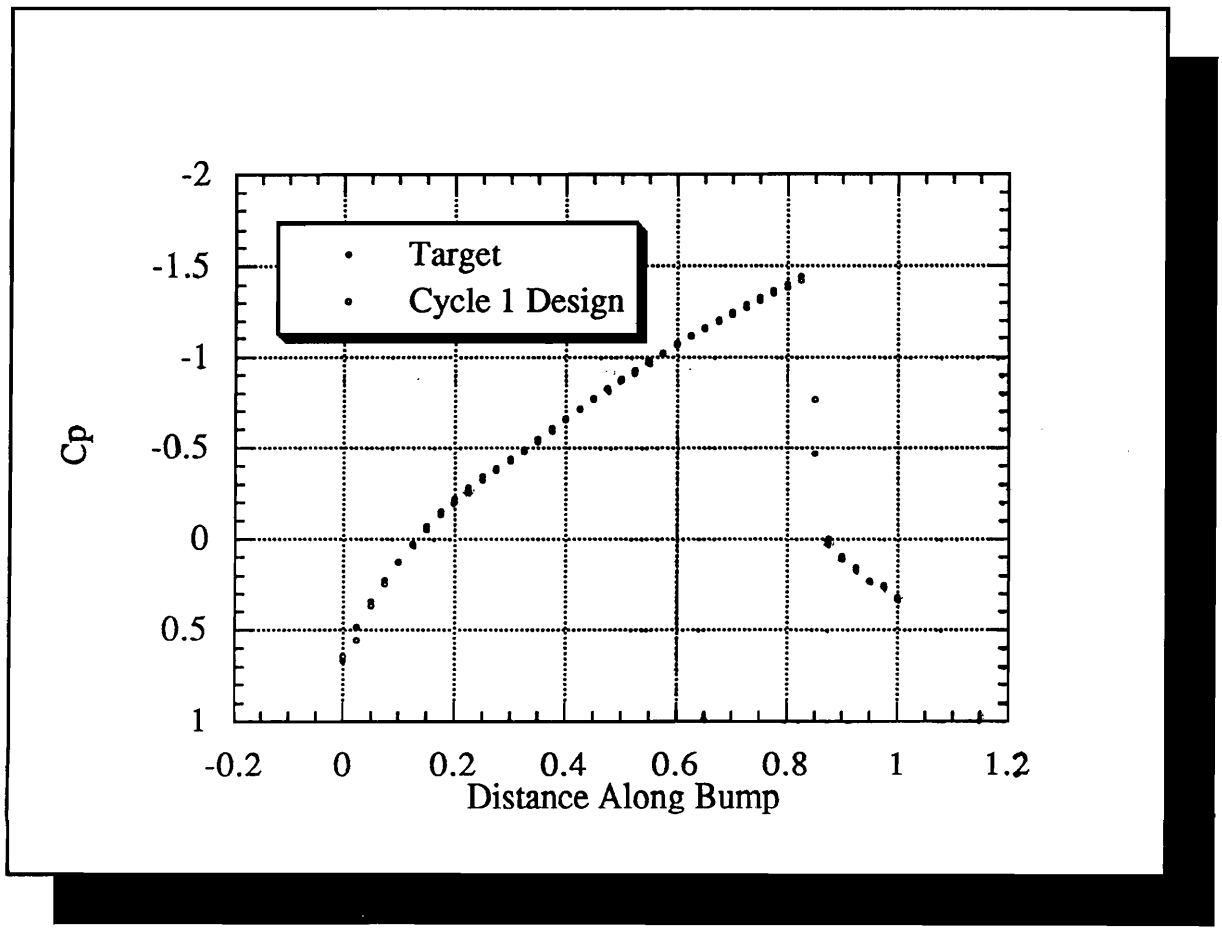

Figure 11.12: Pressure distribution comparison between the first response cycle design and the target.

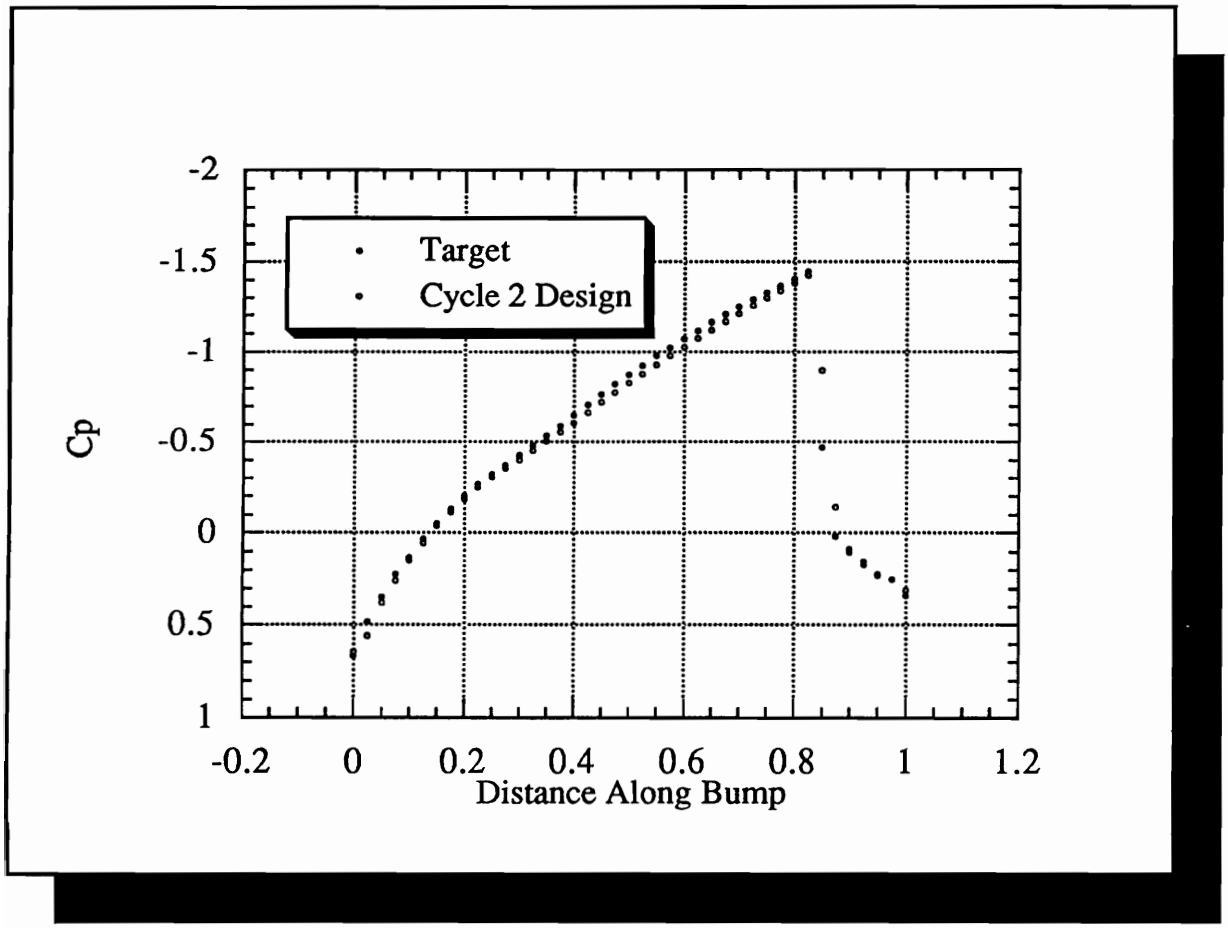

Figure 11.13: Pressure distribution comparison between the second response cycle design and the target.

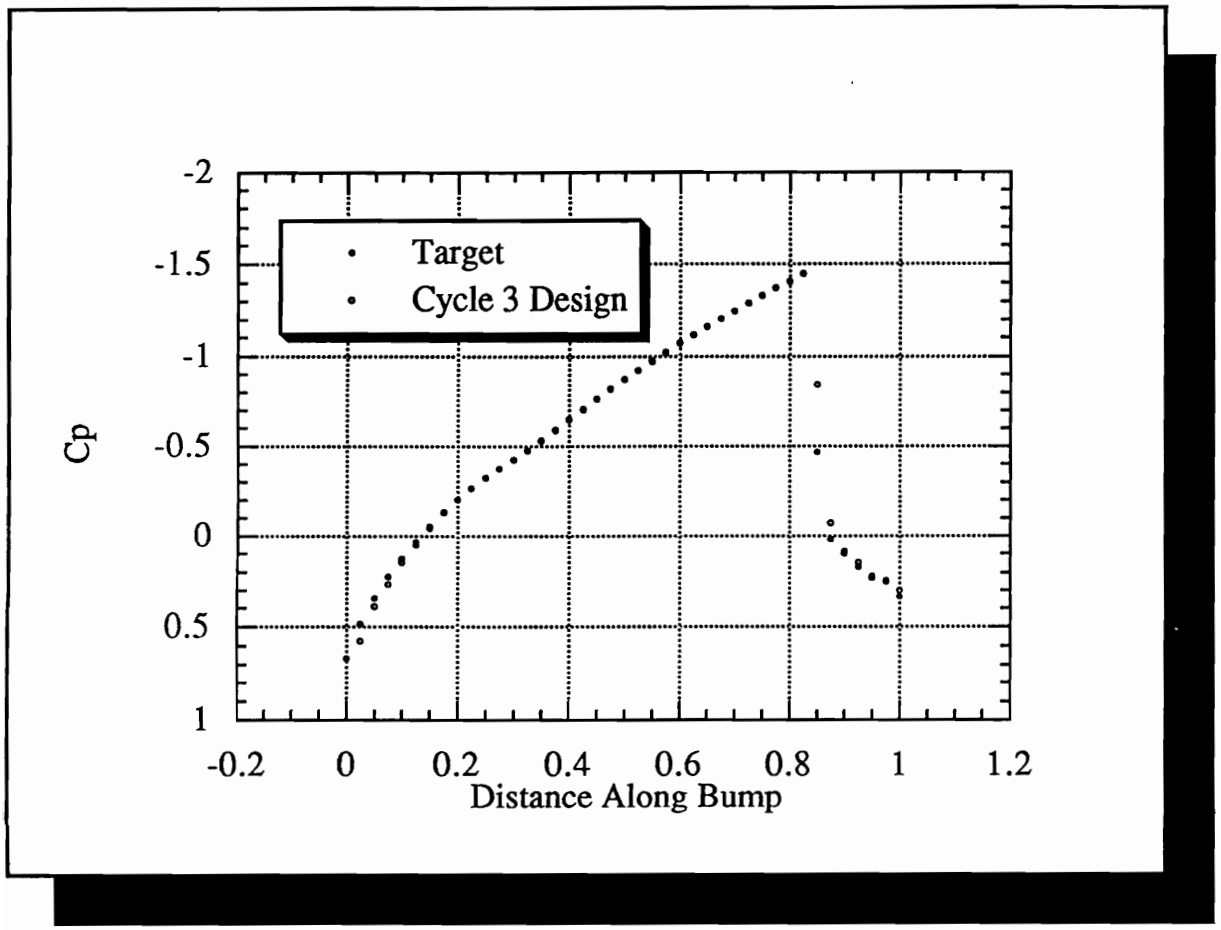

Figure 11.14: Pressure distribution comparison between the third response cycle design and the target.

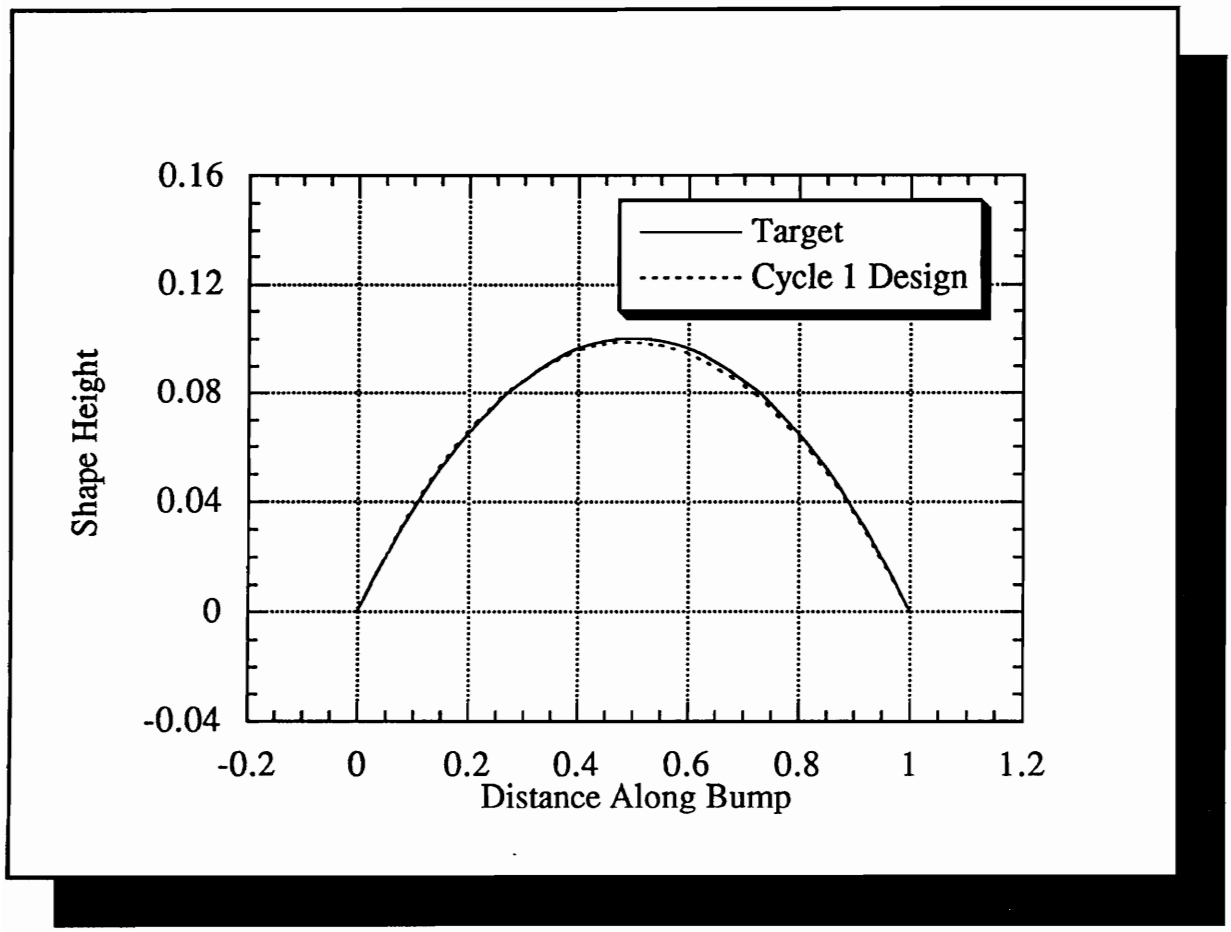

Figure 11.15: Design of the bump in the transonic channel flow after 1 response surface cycle.

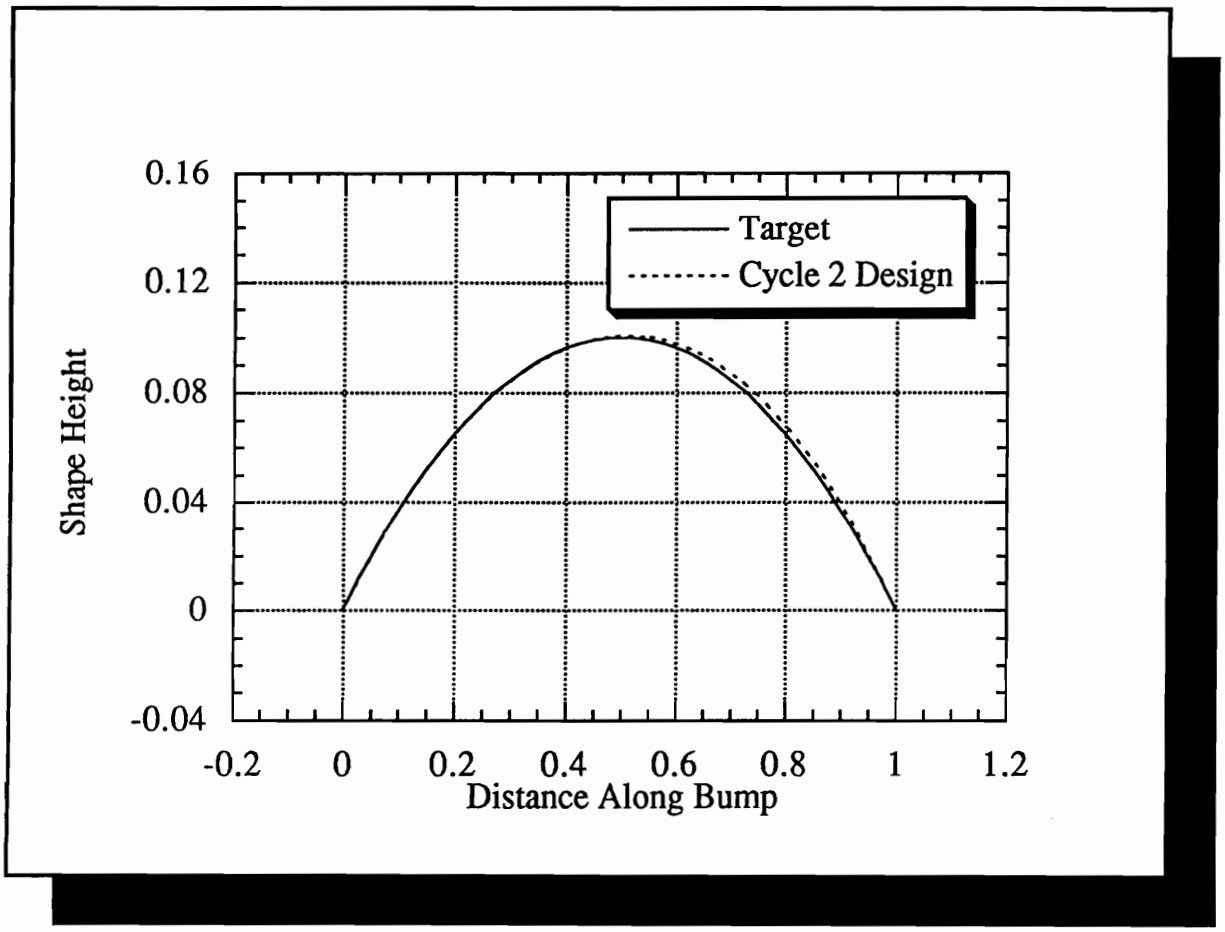

Figure 11.16: Design of the bump in the transonic channel flow after 2 response surface cycles.

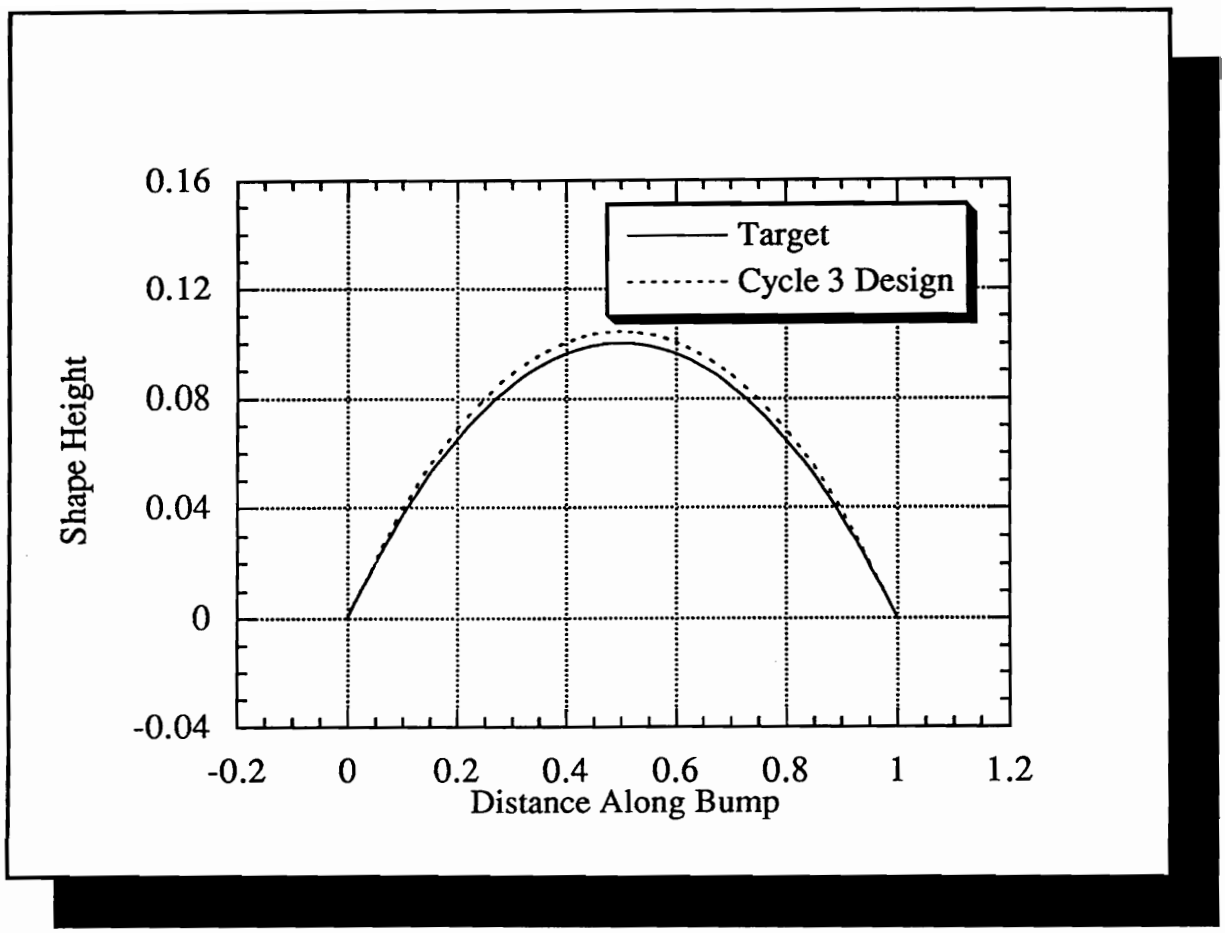

Figure 11.17: Design of the bump in the transonic channel flow after 3 response surface cycles.

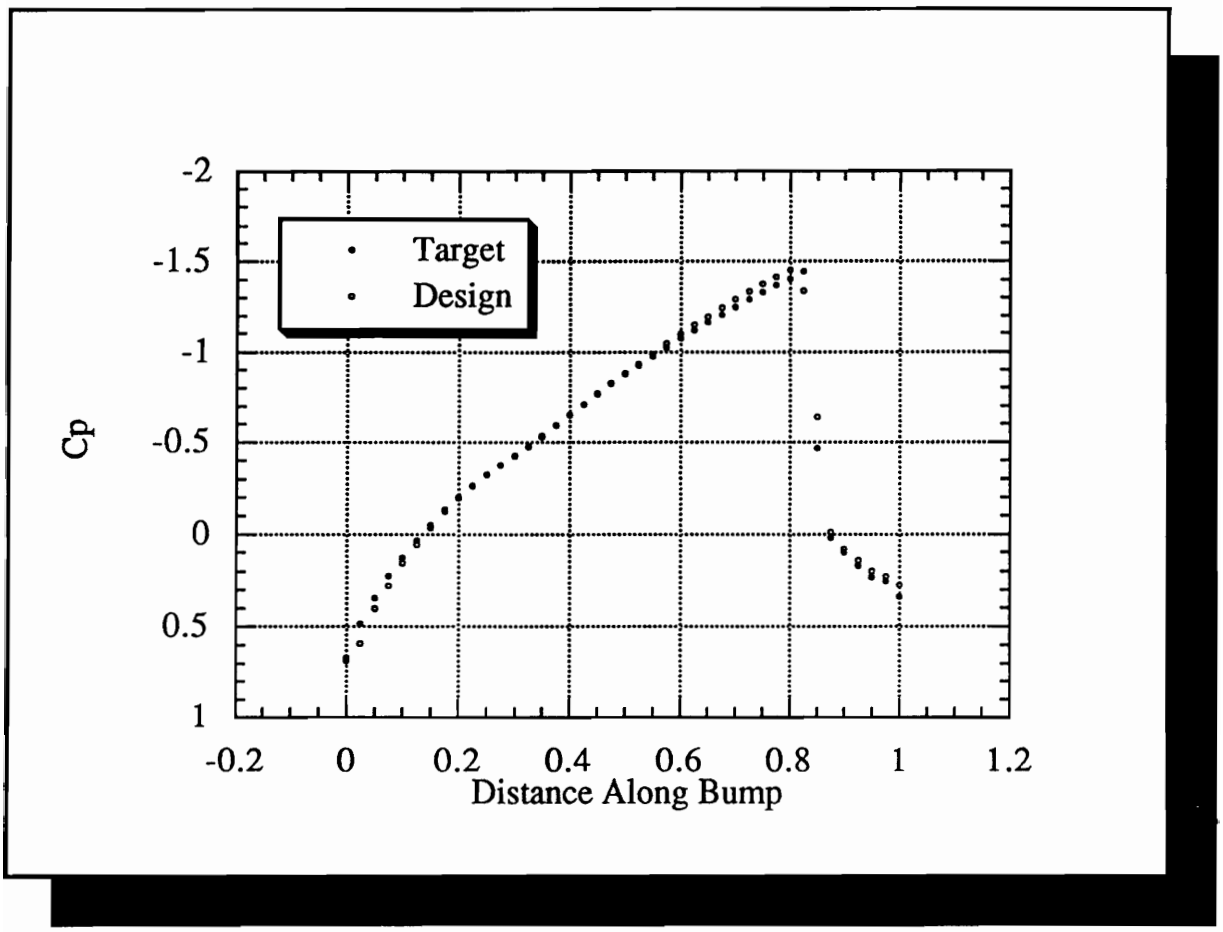

Figure 11.18: Pressure distribution comparison between the design with the lowest objective function encountered during the response surface optimization and the target.

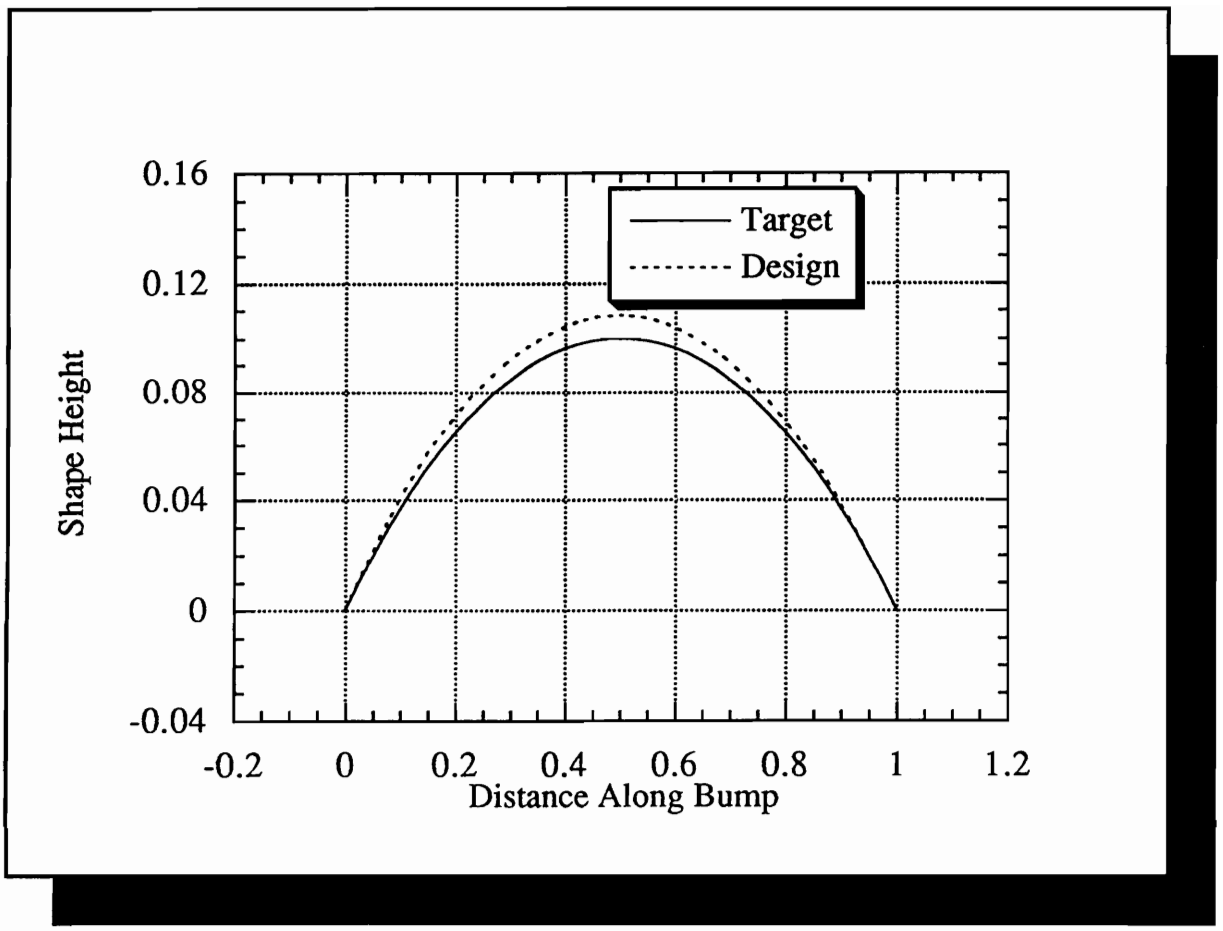

Figure 11.19: Design with the lowest objective function value encountered during the response surface optimizaion of the transonic bump problem.

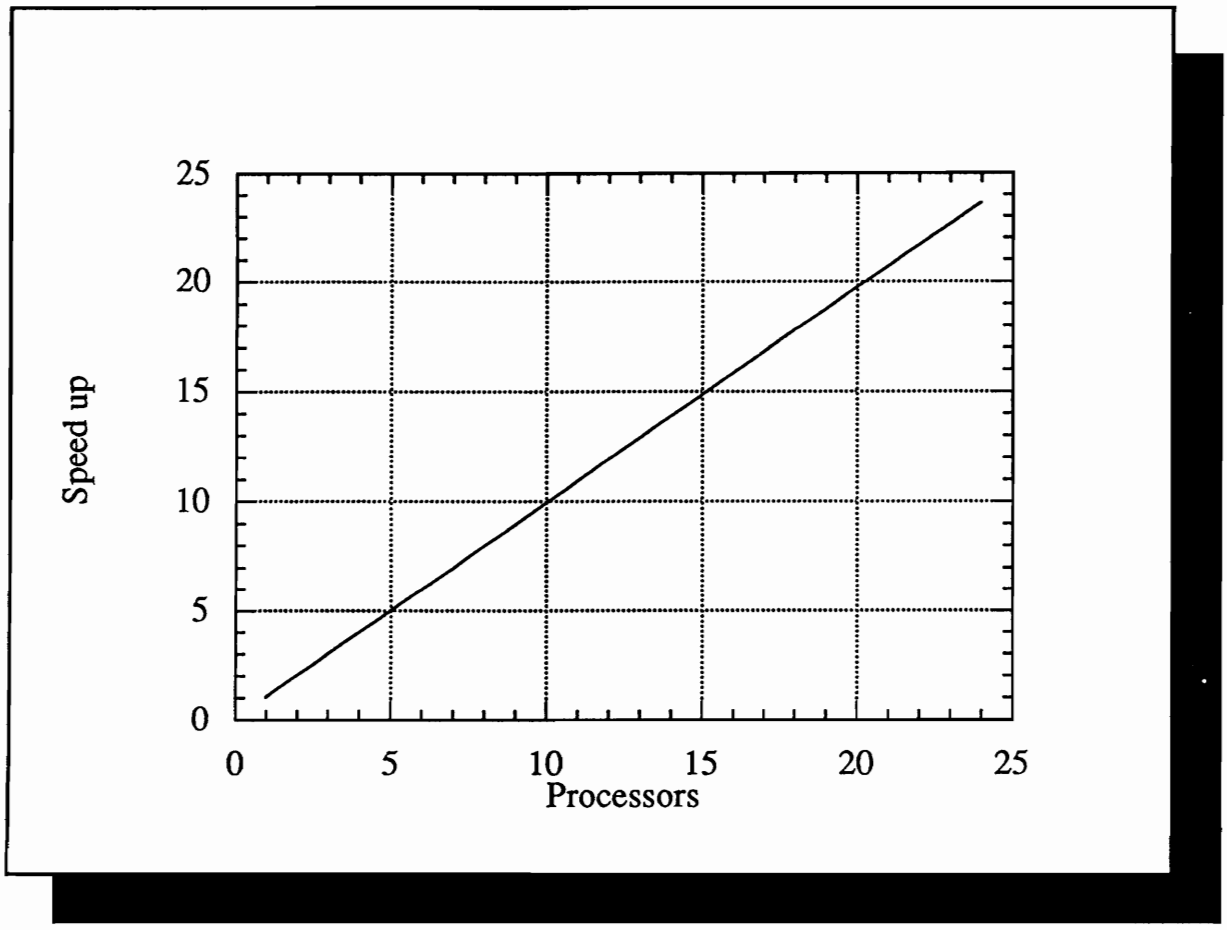

Figure 11.20: Speed-up with parallel computing of Euler solutions to the transonic bump problem for response surface construction.

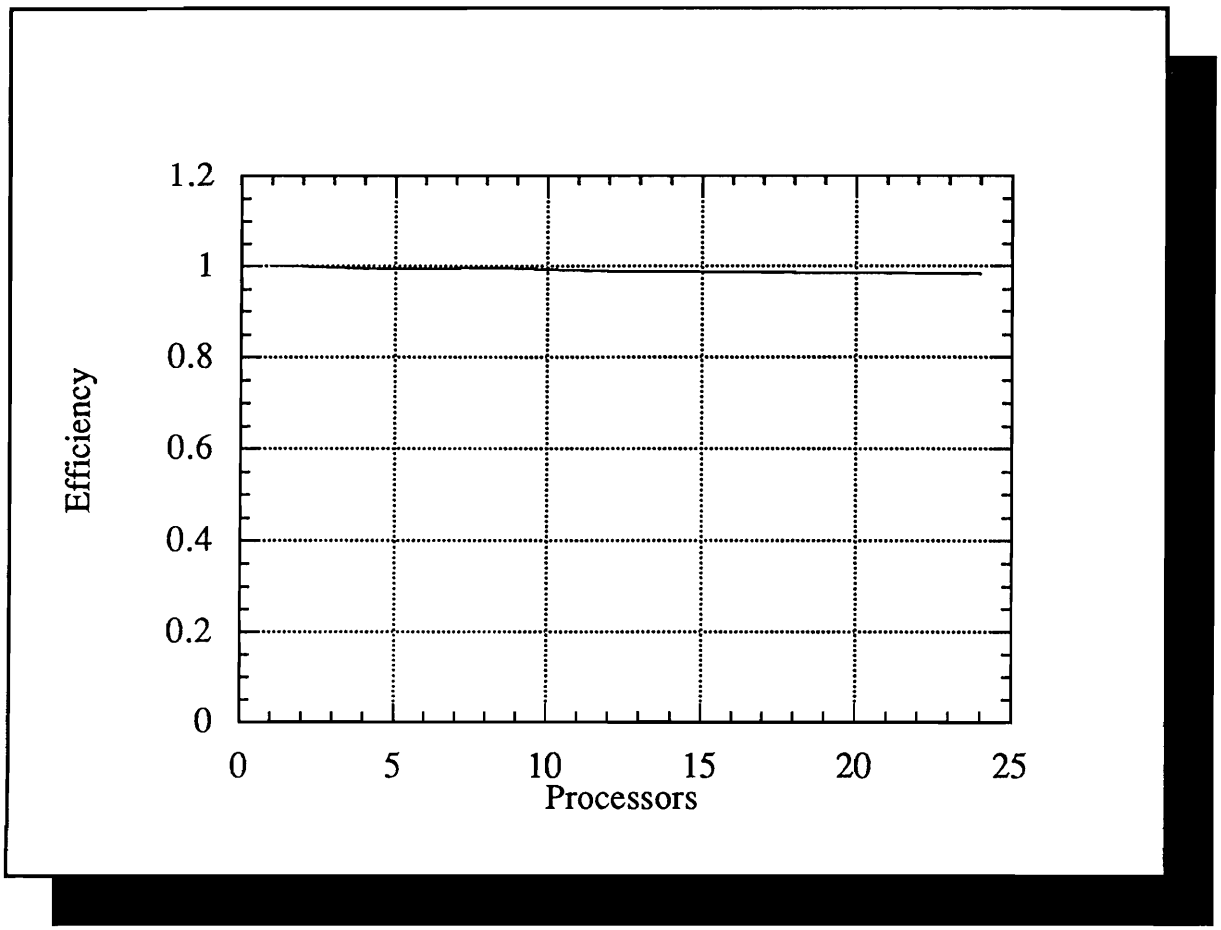

Figure 11.21: Efficiency with parallel computing of Euler solutions to the transonic bump problem for response surface construction.

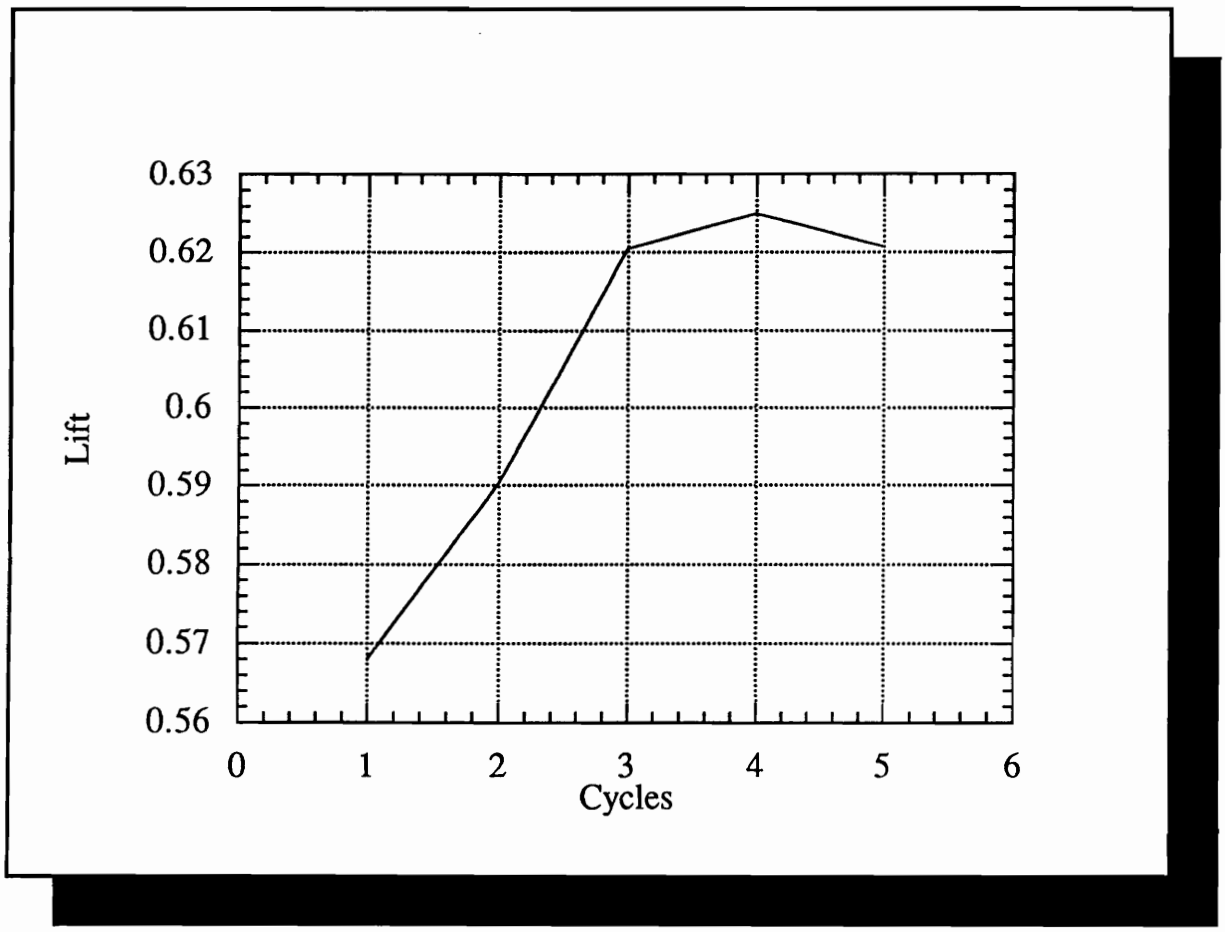

Figure 11.22: Convergence of the transonic airfoil design. Lift is computed at $M = 0.75$ and $\alpha = 0^{\circ}$.

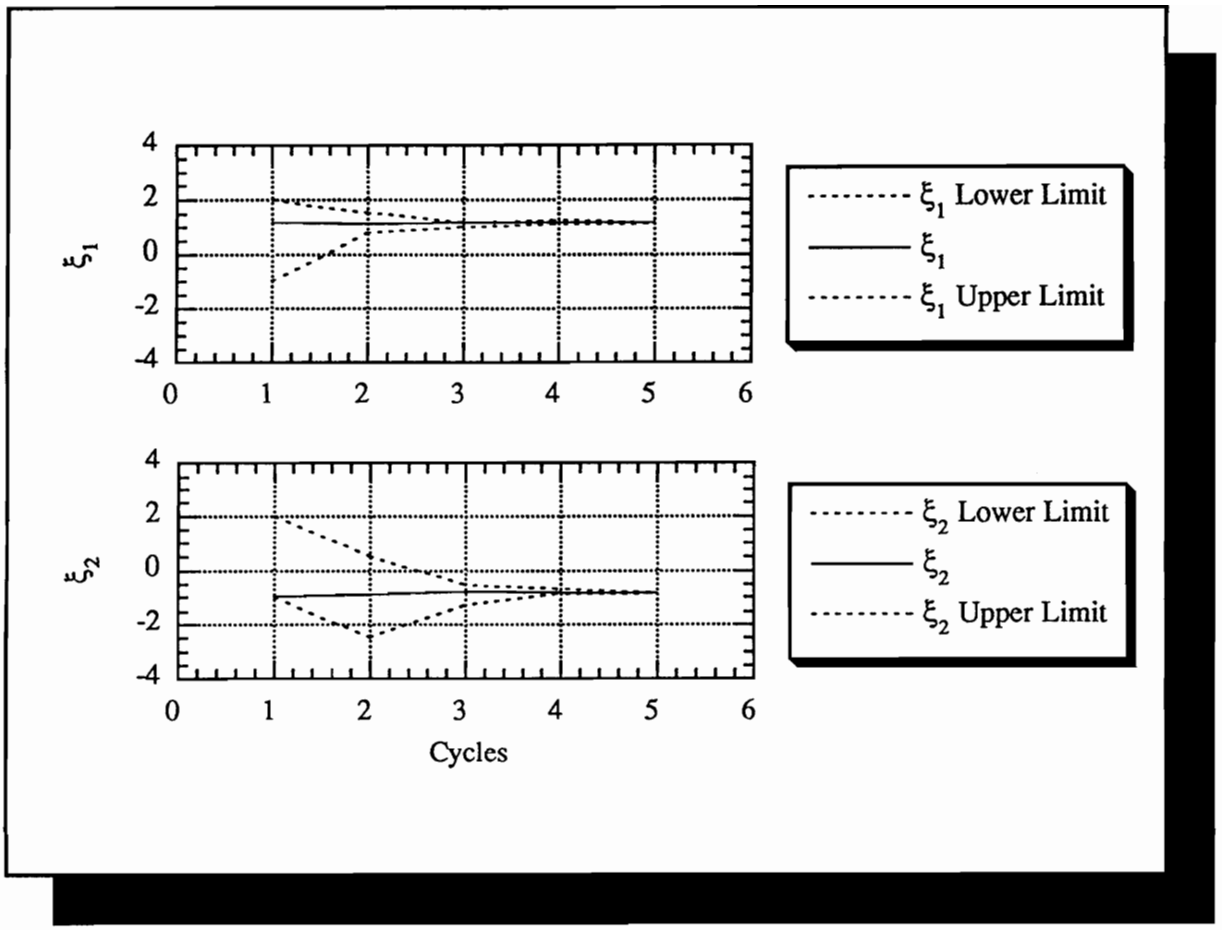

Figure 11.23: Convergence of ξ_1 and ξ_2 for the transonic airfoil design optimized tby response surfaces.

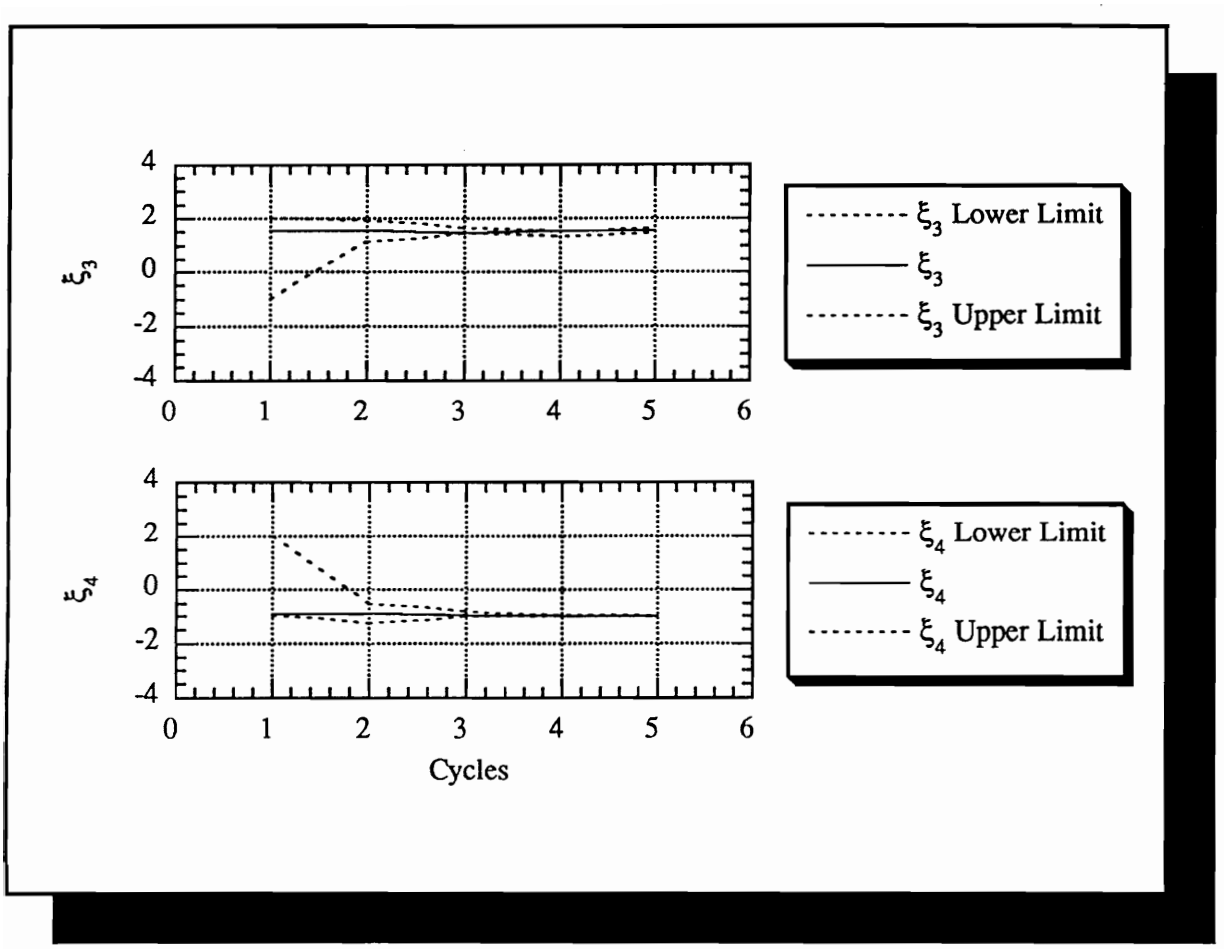

Figure 11.24: Convergence of ξ_3 and ξ_4 for the transonic airfoil design optimized tby response surfaces.

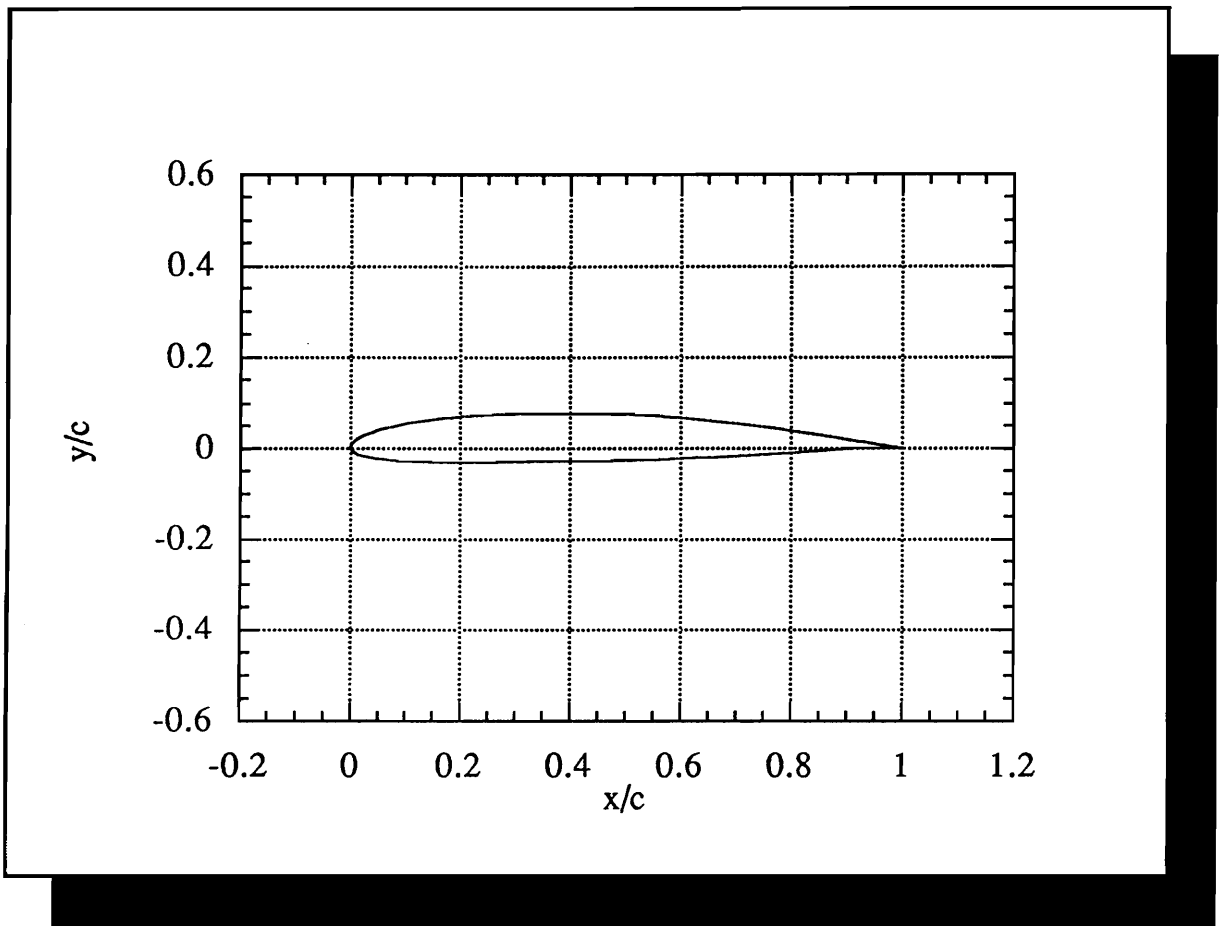

Figure 11.25: Optimized shape for an airfoil at M = 0.75, $\alpha = 0°$.

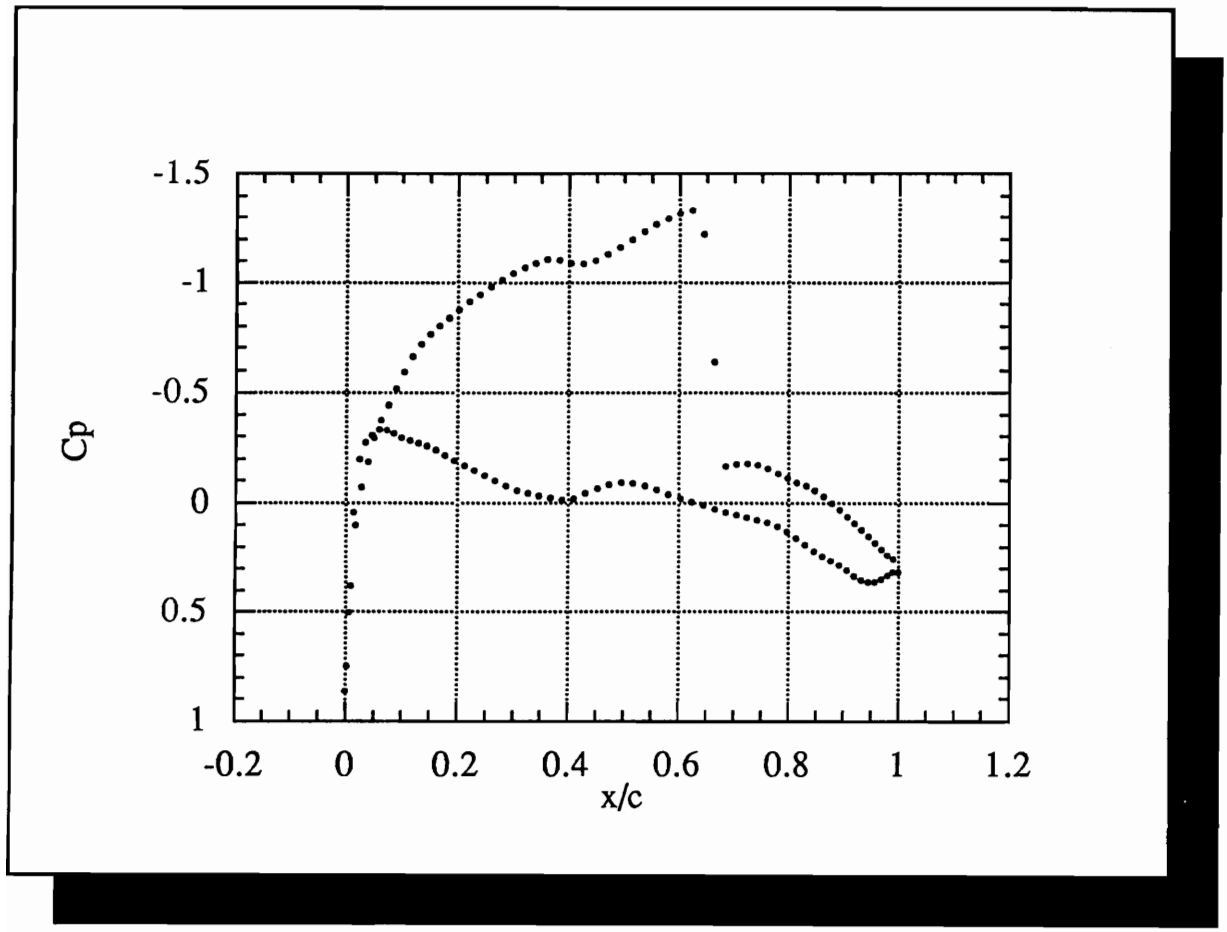

Figure 11.26: Surface pressure distribution for optimized airfoil at M = 0.75, $\alpha = 0°$.

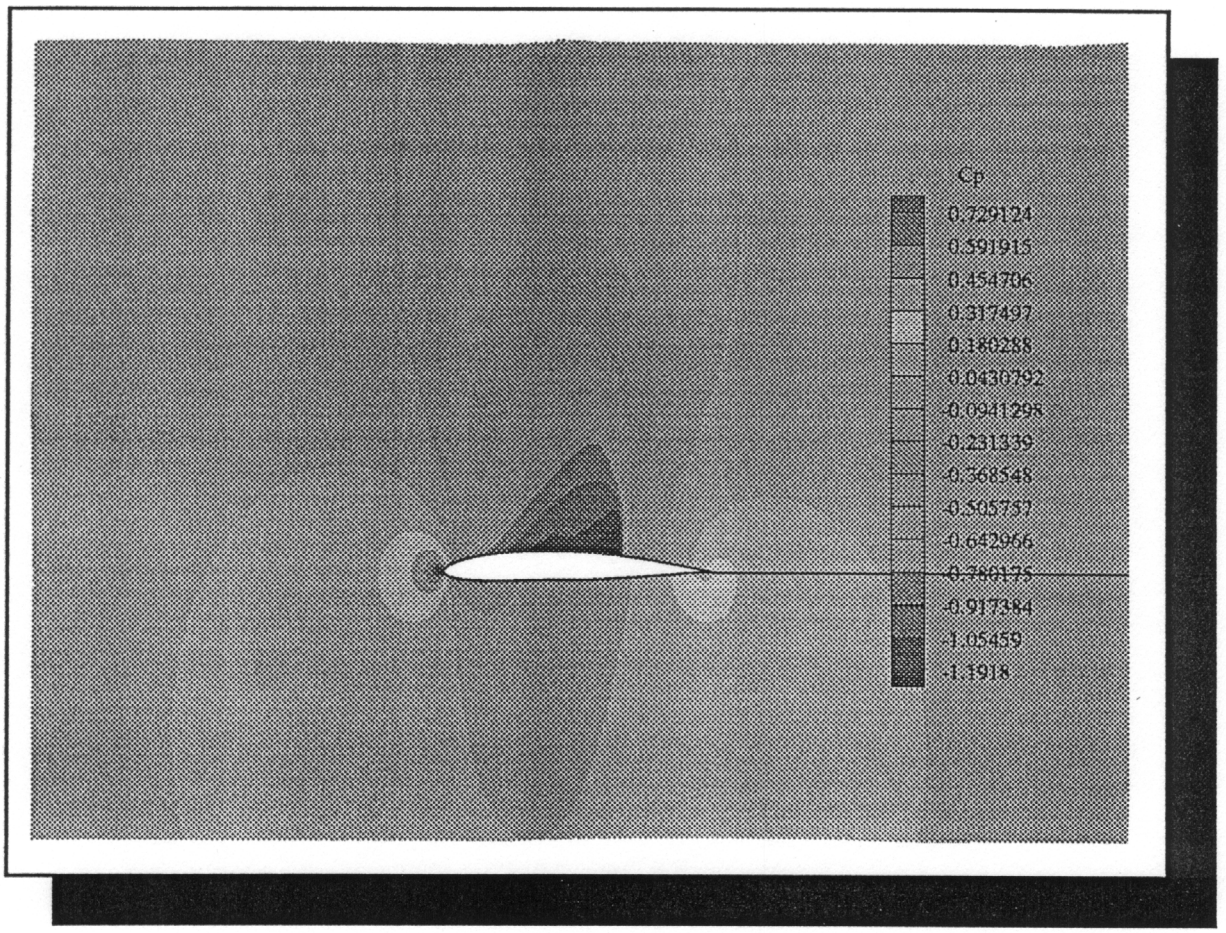

Figure 11.27: Pressure contours in flow field for optimized airfoil at M = 0.75, $\alpha = 0°$.

APPENDIX
A

GRAMPS USER'S GUIDE

This appendix provides some instructions for using GRAMPS, the Genetic Recombination Algorithm for Multiple Point Selection. The input decks are explained fully and some suggestions for parameter values are given. We recommend that the example problems from the chapter 9 be check to make sure the code is running properly. Due to the random function calls and the differences in compilers, recovering the D-optimal results in exactly the reported number of generations is not likely, however, the D-optimal solution should be recovered.

The information which allows GRAMPS to run different problems is read from a series of input files. The main input deck contains all the information specific to setting the GA parameters. The secondary file contains the data to describe the region over which the D-optimal points are to be found. An example of the input is shown in below.

GRAMPSv1 INPUT DECK
written by R. Narducci
August, 1994

Genetic Algorithm Parameters

Population Size	No. of Gen.	Total No. of A^TA Evals.
5	10000	-1
Mutation Rate	Random Number Seed	
0.15	1001	

Curve Fit Parameters

No. of Design Dim	Type of Curve	No. of Pts in Fit	No. of Coef.
7	1	54	36

Design Space Description		
Cubic Space (0/1)	Pts/Limits (0/1)	File Name
0	1	limit.dek
Output specification		
Print Option	File of D-opt Pts	Convergence History
0	dv7c.pts	dv7c.his

To initiate GRAMPS in UNIX-based environments, type

$$gramps < filename$$

where *filename* is the name of the main input deck.

The input deck is entered as a formatted read, therefore it is critical that the information stored in it is aligned correctly. Information is read in columns of 20 spaces wide. Each new piece of data should begin in column 1, 21, 41, or 61. The information is read in 5 cards.

The first two cards contain the GA parameters. The first card contains the population size, total number of generations and the maximum allowable evaluations of $|A^T A|$. The population size for most problems should be in the order of 5 to 10. Rarely should a problem contain more than 30 as large number of designs in the population increases the chances of retaining large amounts of poor designs. The GA will stop after the maximum number of generations has been reached or the limit of $|A^T A|$ evaluations have been reached. The second card sets the mutation rate and the random number seed. The mutation rate is the probability of a gene in a string being mutated. As a rough rule of thumb, this number should be between 5-15%. The random number seed initializes the random number generator. Several starts of the GA can be done for a specific problem by varying the random number seed. This value should be a large, odd integer.

The next card sets the conditions of the GA for a desired response surface type. The first number on the card tell the GA how many dimensions the design space spans. The second number is a flag representing the desired type of response surface. GRAMPS is preprogrammed for the following types of curves:

0.) $f = c_1\xi_1 + c_2\xi_2 + \ldots + c_n\xi_n + c_{n+1}$

1.) $f = c_{11}\xi_1^2 + c_{12}\xi_1\xi_2 + \ldots + c_{1n}\xi_1\xi_n + c_{22}\xi_2^2 + \ldots + c_{2n}\xi_2\xi_n$

$c_1\xi_1 + c_2\xi_2 + \ldots + c_n\xi_n + c_{n+1}$

2.) $f = (c_1\xi_1 + c_2)(c_3\xi_1 + c_4)$ *or*

$f = (c_1\xi_1 + c_2)(c_3\xi_1 + c_4)(c_5\xi_1 + c_6)$

3.) $f = (c_1\xi_1^2 + c_2\xi_1 + c_3)(c_4\xi_1^2 + c_5\xi_1 + c_6)$ *or*

$f = (c_1\xi_1^2 + c_2\xi_1 + c_3)(c_4\xi_1^2 + c_5\xi_1 + c_6)(c_7\xi_1^2 + c_8\xi_1 + c_9)$

4.) $f = (c_1\xi_1 + c_2)(c_3\xi_1^2 + c_4\xi_1 + c_5)$

The next number represents the number of points wanted in the least-squares problem. The last number is the number of coefficients in the curve. This number must be less than or equals to the previous.

The third card gives information pertaining to the description of the design space. The first number flags whether the space is a hypercube or not. In other words, if the flag equal 0 the genes will be represented as coordinates of the points in the design space or otherwise if equal to 1 the genes will be represented by numbers assigned to the points. There are two ways to input the points discretizing the design space. Both ways need another data file. The first way (assign the flag to 0) is by giving a file with a list of points. The second way is to give the lower and upper limits of the design space in each direction. More will be given on the format of the data file later.

The final card sets the output option. The first is the print option to the screen. Set equal to zero, all output to the screen is suppressed. Set equal to one, a history

of the max$|A^TA|$ and ave$|A^TA|$ for the population scrolls through the screen for each generation. Set equal to two, gives a description of all designs in a population, and the parents the design came from. This is a good option for visualizing the progress of the GA. The next two pieces of information give the output file names. The first file stores the D-optimal points; the second stores a history of the maximum and average $|A^TA|$.

The file containing the design space description also has some specific formats. When the design space is given as a set of limits, then the information is given in a file as shown below

GRAMPSv1 Design Space Limits Deck
written by R. Narducci
August, 1994

Lower Limit	Upper Limit	No. of Levels
1.00	1.10	5
1.10	1.20	5
1.20	1.30	5
1.30	1.40	5
1.40	1.50	5
1.50	1.60	5
1.60	1.70	5

The first column shows the lower limit, followed by the upper limit, followed by the the number of points in the discretization of that direction. If the description of the space is not a hypercube type then a list of points must be given. The first line of the data file is the number if points representing the design space. The list of points begins on line two. The coordinate of the points should appear in columns. At the end of each line and integer should appear. If the integer is 1, then the point will be selected as a D-optimal point. If the integer is 0, then the point may of may not be selected for curve fitting. Note that if points are preselected, the set of points may not be truly D-optimal.

APPENDIX
B

GRAMPS FORTRAN CODE

The following is a listing of the genetic algorithm GRAMPS or Genetic Recombination Algorithm for Multiple Point Selection. Several subroutines are omitted because they are lengthy and readily available in other sources or they have copyright protection. Of those missing are the LAPACK subroutine dgeqr2.f which performs QR factorization on a matrix, and sort.f which is found in Ref. 70 and performs a ranking of an array of numbers.

```
      program main

c     ------------------------------------------------------------
c     Front end for the Genetic Recombination Algorithm code for
c     Multiple Point Selection code.        Last Update: 8-26-94
c     ------------------------------------------------------------

c     -----------------
c     Type declarations
c     -----------------
      integer piw, pw
      parameter(piw = 232)
      parameter(pw  = 1292)

      integer ndv, ifit, npts, npop, ngen, npool, iprint
      integer igenes, ncoef, ireal8, iint, iw(piw), nfcn, ispace
      integer pipar1, pipar2, pipsp, piord, pitmp, pxpool, px, pxp
      integer pxtemp, pxline, pd, pp, ppop, pkid, ptmp

      real*8 w(pw)

      character*30 convhis, ptsfile, desspace

c     ---------------
c     Read Input Deck
c     ---------------
      call input(npop, ngen, nfcn, ndv, ifit, npts, ispace, npool,
     .     ncoef, iprint, igenes, desspace, ptsfile, convhis)
```

```
c     -----------------------
c     Check Memory Requirement
c     -----------------------
      call calcmem(npop, npool, ndv, npts, ncoef, igenes, iint, ireal8)
      if ( (iint .gt. piw) .or. (ireal8 .gt. pw) ) then
        write(*,902) '*******************************************'
        write(*,901) 'The parameter piw must be at least:', iint
        write(*,901) 'The parameter pw must be at least: ', ireal8
        write(*,902) 'Adjust these parameters in main.f subroutine'
        write(*,902) '*******************************************'
        stop
      end if

c     -------------------------
c     Assign Pointers, Integers
c     -------------------------
      pipar1 = 1
      pipar2 = pipar1 + npop
      pipsp  = pipar2 + npop
      piord  = pipsp + npool
      pitmp  = piord + npop

c     ----------------------
c     Assign Pointers, Real*8
c     ----------------------
      pxpool = 1
      px = pxpool + npool*ndv
      pxp = px + npts*ncoef
      pd = pxp + (npts-igenes/ndv)*ncoef
      pp = pd + npop
      ppop = pp + npop
      pkid = ppop + npop*igenes
      pxtemp = pkid + npop*igenes
      pxline = pxtemp + ndv
      ptmp = pxline + ncoef

      call gramps(npop, ngen, nfcn, ndv, ifit, npts, ispace, npool,
     .     ncoef, iprint, igenes, desspace, ptsfile, convhis,
     .     iw(pipar1), iw(pipar2), iw(pipsp), iw(piord), w(pxpool),
     .     w(px), w(pxp), w(pd), w(pp), w(ppop), w(pkid), w(pxtemp),
     .     w(pxline), w(ptmp))

901   format (a,i9)
902   format(a)
      end

c-----------------------------------------------------------------------
      subroutine input(npop, ngen, nfcn, iseed, ndv, ifit, npts, ispace,
     .           ifile, ncoef, iprint, xmut, desspace, ptsfile, convhis)

c     This subroutine reads the input deck for GRAMPSv1
c
c     -------------------
c     Variable Definitions
c     -------------------
```

```
c     ---------------
c     Input variables
c     ---------------
c     none
c
c     ----------------
c     Output variables
c     ----------------
c     npop              No. of designs in a single generation (population)
c     ngen              No. of generations to run GA
c     nfcn              Maximum No. of function evaluation to be done in GA
c     xmut              Mutation rate
c     iseed             Random number seed
c     ndv               No. of dimensions in design space
c     ifit              Type of function to describe the response surface
c     npts              No. of points to be selected for D-optimality
c     ncoef             No. of coefficients in polynomial used to describe
c                         the response surface
c     ispace            Flag describing design space
c     ifile             Flag for design space representation
c     desspace          File containing design space description
c     iprint            Print option
c     ptsfile           File to contain D-optimal points
c     convhis           File to contain convergence history
c
c     -----------------
c     Working Variables
c     -----------------
c     temp
c-----------------------------------------------------------------------
c     --------------
c     Variable Block
c     --------------
      integer npop, ngen, nfcn, iseed, ndv, ifit, npts, ncoef
      integer ispace, ifile, iprint

      real*8 x, xmut

      character*30 desspace, ptsfile, convhis
      character*72 temp

c     --------------
c     Reading Header
c     --------------
      open (2, file = 'gramps.dek', status = 'old')
      read (2,901) temp
      write(*,901) temp
      read (2,901) temp
      write(*,901) temp
      read (2,901) temp
      write(*,901) temp
      read (2,901) temp
      write(*,901) temp

c     ---------------------
```

```
c     Reading GA Parameters
c     ---------------------
      read (2,901) temp
      write(*,901) temp
      read (2,901) temp
      write(*,901) temp
      read (2,901) temp
      write(*,901) temp
      read (2,902) npop, ngen, nfcn
      write(*,903) npop, ngen, nfcn
      read (2,901) temp
      write(*,901) temp
      read (2,911) xmut, iseed
      write(*,912) xmut, iseed

c     ----------------------------
c     Reading Curve Fit Parameters
c     ----------------------------
      read (2,901) temp
      write(*,901) temp
      read (2,901) temp
      write(*,901) temp
      read (2,901) temp
      write(*,901) temp
      read (2,904) ndv, ifit, npts, ncoef
      write(*,905) ndv, ifit, npts, ncoef

c     -------------------------------
c     Reading Design Space Description
c     -------------------------------
      read (2,901) temp
      write(*,901) temp
      read (2,901) temp
      write(*,901) temp
      read (2,901) temp
      write(*,901) temp
      read (2,906) ispace, ifile, desspace
      write(*,907) ispace, ifile, desspace

c     ---------------------------------
c     Reading Output File Specifications
c     ---------------------------------
      read (2,901) temp
      write(*,901) temp
      read (2,901) temp
      write(*,901) temp
      read (2,901) temp
      write(*,901) temp
      read (2,908) iprint, ptsfile, convhis
      write(*,909) iprint, ptsfile, convhis
      read (2,901) temp
      write(*,910) temp

      close(2)
```

```
c     -----------------
c     Format Statements
c     -----------------
901   format(a)
902   format(3(i20))
903   format(3(i10,10x))
904   format(4(i20))
905   format(3(i10,10x),i5)
906   format(2(i20),a20)
907   format(2(i10,10x),a20)
908   format(i20,a20,a20)
909   format(i10,10x,a20,a20)
910   format(a,//)
911   format(f20.10,i20)
912   format(f10.4,10x,i10)

      return
      end

c-----------------------------------------------------------------------
      subroutine limits(ifile, ndv, nlim, nxpool, nipsp, ilim, ipsp,
     .           ips, npool, xl, xu, xpool, despace)

c     This subroutine reads in the description of the design space or
c     computes it based on information from despace.
c
c     --------------------
c     Variable Description
c     --------------------
c
c     ---------------
c     Input Variables
c     ---------------
c     ifile             Flag for design space representation
c     ndv               No. of dimensions in design space
c     nlim              Length of the arrays ilim, xl, xu
c     nxpool            Dimension of array xpool
c     nipsp             Dimension of array ipsp
c     despace           File containing design space description
c
c     ----------------
c     Output Variables
c     ----------------
c     ipsp              Array identifying preselected points
c     ips               No. of preselected points
c     npool             No. of points to pick D-optimal set from
c     ilim              Array of size (nlim) no. of discrete pts along the
c                         ith design dimension
c     xl                Array of size (nlim), lower limit of variables
c     xu                Array of size (nlim), upper limit of variables
c     xpool             Array of size (nxpool) containing description of
c                         points in design space
c
c     -------------
c     Working Array
```

```
c     -------------
c     i, j              Counters
c     temp              Character temporary array
c
c-----------------------------------------------------------------------
c     -------------
c     Variable Block
c     -------------
      integer i, j, ifile, ndv, nxpool, nipsp, ilim(nlim), ipsp(nipsp)
      integer ips, npool, nlim

      real*8 xl(nlim), xu(nlim), xpool(nxpool)

      character*30 despace
      character*72 temp

c     --------------------
c     Initialize subroutine
c     --------------------
      open (3, file = despace, status = 'old')
      ips = 0

c     ------------------------------
c     Set up design space description
c     ------------------------------
      if (ifile .eq. 0) then
        read(3,*) npool
        do 10 j = 1, npool
          read(3,*) (xpool((i-1)*npool+j), i = 1, ndv), ipsp(j)
          ips = ips + ipsp(j)
10      continue
      else
        read(3,901) temp
        read(3,901) temp
        read(3,901) temp
        read(3,901) temp
        read(3,901) temp
        read(3,901) temp
        read(3,901) temp
        do 20 i = 1, ndv
          read(3,*) xl(i), xu(i), ilim(i)
20      continue
        do 30 i = 1, ndv
          do 30 j = 0, ilim(i)-1
            xpool(j*ndv+i) = xl(i) - real(j)*(xl(i) - xu(i))/
              real(ilim(i)-1)
30      continue
        npool = 1
        do 40 i = 1, ndv
          npool = npool*ilim(i)
40      continue
      end if

      close(3)

901   format(a)
```

```
      return
      end

c-----------------------------------------------------------------------
      subroutine calcmem(ifile, npop, npool, ndv, npts, ncoef, ngenes,
     .           nlim, nxpool, nipsp, nint, nreal, despace)

c     This subroutines computes the necessary storage space to be
c     allocated for GRAMPS.  This routine also computes the number
c     of genes to represent thD-optimal designs
c
c     --------------------
c     Variable definitions
c     --------------------
c
c     input variables
c     ---------------
c     ifile             Flag for design space representation
c     npop              No. of designs in a single generation (poplulation)
c     npool             No. of points describing entire design space
c     ndv               No. of dimensions in design space
c     npts              No. of points to be selected for D-optimality
c     ncoef             No. of coefficients in polynomial used to describe
c                         the response surface
c     ngenes            No. of genes representing each design
c     despace           File containing design space description
c
c     output variables
c     ----------------
c     nlim              Length of the arrays ilim, xl, xu
c     nxpool            Length of the array xpool
c     nipsp             Length of the array ipsp
c     nint              Amount of space to be allocated for integers
c     nreal             Amount of space to be allocated for real*8
c
c     working variables
c     -----------------
c     i, j              counters
c     ctemp             temporary character string
c     xtemp             temporary real*8
c     itemp             No. of discretized points in one design dimension
c     max               Max no. of discretized points in a single design
c                         dimension
c-----------------------------------------------------------------------
c     --------------
c     Variable Block
c     --------------
      integer ifile, npop, npool, ndv, npts, ncoef, ngenes
      integer nlim, nxpool, nipsp, nint, nreal
      integer i, j, itemp, max

      real*8 xtemp

      character*30 despace
      character*72 ctemp
```

```
c     ------------------------------------------------------
c     Compute length of arrays ilim, xl, xu, xpool, and ipsp
c     ------------------------------------------------------
      open(3, file = despace, status = 'old')
      if (ifile .eq. 0) then
        read(3,*) npool
        nlim   = 1
        nxpool = npool*ndv
        nipsp  = npool
      else
        max = 0
        read(3,901) ctemp
        read(3,901) ctemp
        read(3,901) ctemp
        read(3,901) ctemp
        read(3,901) ctemp
        read(3,901) ctemp
        do 10 i = 1, ndv
          read(3,*) xtemp, xtemp, itemp
          if (itemp .gt. max) then
            max = itemp
          end if
10      continue
        nlim   = ndv
        nxpool = max*ndv
        nipsp  = 1
      end if

c     ----------------------
c     Compute number of Genes
c     ----------------------
      ngenes = 0
      if (ifile .eq. 0) then
        do 20 i = 1, npool
          read(3,*) (xtemp, j = 1, ndv), itemp
          ngenes = ngenes + itemp
20      continue
          ngenes = (npts - ngenes)*ndv
      else
        ngenes = npts*ndv
      end if
      close(3)

c     ---------------------
c     Compute integer memory
c     ---------------------
      nint = 1 + nlim + 3*npop + nipsp + npool

c     --------------------
c     Compute real*8 memory
c     --------------------
      nreal = ncoef*(2*npts - ngenes/ndv + 3) + 2*npop*(1 + ngenes) +
     .        ndv + nxpool + npool + 2*nlim + 2
c     nreal = 100*nreal + 1000
```

```
 901  format(a)
      return
      end

c-----------------------------------------------------------------------
      subroutine gramps(npop, ngen, nfcn, iseed, ndv, ifit, npts,
     .           ispace, npool, nlim, nxpool, nipsp, ncoef, iprint,
     .           ig, ips, ifile, xmut, idim, ptsfile, convhis, ipar1,
     .           ipar2, ipsp, iord, xpool, x, xp, d, p, pop, kid,
     .           xtemp, xline, ixdis, xdis, w)

c-----------------------------------------------------------------------
c     This program implements a variation of the genetic algorithm
c     discussed in the paper by Furuya and Haftka entitled "Locating
c     Actuators for Vibration Suppression on Space Trusses by Genetic
c     Algorithms", ASME Winter Annual Meeting 1993.  The purpose of
c     the code is to find a set of D-optimal points from a region.
c
c     -------------------
c     Variable Definition
c     -------------------
c     ---------------
c     Input Variables
c     ---------------
c     npop              No. of designs in a single generation (population)
c     ngen              No. of generations to run GA
c     nfcn              Maximum No. of function evaluations to be done in GA
c     iseed             Random number generator seed
c     ndv               No. of dimensions in design space
c     ifit              Type of function to describe the response surface
c     npts              No. of points to be selected for D-optimality
c     ispace            Flag describing design space
c     npool             No. of points describing entire design space
c     nlim              Length of array ilim
c     nxpool            Length of array xpool
c     nipsp             Length of array ipsp
c     ncoef             No. of coefficients in polynomial used to describe
c                         the response surface
c     iprint            Print option
c     ig                No. of genes representing each design
c     ips               No. of preselected points
c     ifile             Flag for design space representation
c     xmut               Mutation rate
c     idim              Array of dimension (nlim); No. of discrete pts
c                         along the ith design dimension
c     ptsfile           File to contain D-optimal points
c     conhis            File to contain convergence history
c     xpool             Array of dimension nxpool; contains points describing
c                         design space
c
c     -----------------
c     Working Variables
c     -----------------
c     i, j, k, n,       general counters
```

```
c       kk, jj
c     cc                counter for numbering coefficients
c     igen              counter for numbering generation
c     ir                random integer
c     xr                random real*8
c     rand              random number function
c     ipar1             Array of dimension npop; contains 1st parent i.d.
c     ipar2             Array of dimension npop; contains 2nd parent i.d.
c     ipsp              Array of dimension nipsp; contains id for pre-
c                         selected points
c     iord              Array of dimension npop; contains ranking of
c                         population
c     ixdis             Array of dimension npool; contain ranking for
c                         mutation of general shape region
c     xdis              Array of dimension npool; contains distance info
c                         for mutation of general shape region.
c     x                 Array of dimension npts x ncoef; contains LS matrix
c     xp                Array of dimension npts-ig/ndv x ncoef; contains
c                         LS matrix for preselected points
c     d                 Array of dimension npop; determinant of xTx for
c                         each design in the species
c     aved              Average determinant over the population
c     p                 Array of dimension npop; contains probability
c                         function
c     pop               Array of dimension npop x ig describing the current
c                         population
c     kid               Array of dimension npop x ig describing children
c                         designs
c     xtemp             Temporary array of length ndv
c     xline             Array of length ncoef; contains a row of the LS
c                         matrix
c     w                 Work array
c     irank             Temporary integer containing rank
c     info              Temporary integer of LAPACK subroutines
c     denom             Temporary real*8 number
c     xnop              Number of combinations of points to be chosen from
c                         pool
c     pwork             Pointer
c     ptau              Pointer
c     ptmp              Pointer
c
c     ------------------
c     Subroutines Called
c     ------------------
c     evals, srand, rand, getxl, dgeqr2, sort
c
c     Written by Bob Narducci                              Virginia Tech
c                May, 1994                               215 Randolph Hall
c     Modified   Jan, 1995                               Blacksburg, VA 24060
c-----------------------------------------------------------------------

c     --------------
c     Variable Block
c     --------------
      integer i, j, k, n, cc, jj, kk, ir, igen
```

```
      integer npop, ngen, nfcn, iseed, ndv, ifit, npts, ispace, npool
      integer nlim, nxpool, nipsp, ncoef, iprint, ig, ips, ifile
      integer idim(nlim), ipar1(npop), ipar2(npop), ipsp(nipsp)
      integer iord(npop), ixdis(npool), irank, info
      integer pwork, ptau, ptmp

      real*8 xmut, xpool(nxpool), x(npts,ncoef), xp(npts-ig/ndv,ncoef)
      real*8 d(npop), p(npop), aved, pop(npop,ig), kid(npop,ig)
      real*8 xtemp(ndv), xline(ncoef), xdis(npool), rand, xr
      real*8 denom, xnop, w(1)

      character*30 convhis, ptsfile

c     ---------------------------------
c     Initializing Code & Assign Pointers
c     ---------------------------------
      pwork = 1
      ptau  = pwork + ncoef
      ptmp  = ptau + ncoef

      open (3, file = convhis, status = 'unknown')
      open (7, file = ptsfile, status = 'unknown')

c     ---------------------------------
c     Compute No. of Possible Combinations
c     ---------------------------------
      call evals(npool, npts, ips, xnop)

c     -------------------------------
c     Computing Probability Function
c     -------------------------------
      call srand(iseed)
      denom = dreal(npop*(npop + 1))
      p(1) = 0.0d0
      do 20 i = 1, npop-1
        p(i+1) = p(i) + 2.0d0*dreal(npop+1-i)/denom
20    continue

c     ------------------
c     Preselected Points
c     ------------------
      if (ips .ne. 0) then
        n = 0
        do 50 i = 1, npool
          if (ipsp(i) .eq. 1) then
            n = n + 1
            do 30 j = 1, ndv
              xtemp(j) = xpool((j-1)*npool+i)
30          continue
            call getxl(ifit, ncoef, ndv, xtemp, xline)
            do 40 j = 1, ncoef
              xp(n,j) = xline(j)
40          continue
          end if
50      continue
```

```
      end if

c     ------------------------------------------------------
c     Random Selection of Initial Population.  Genes are chosen
c     randomly, but parents are checked against having duplicate
c     points
c     ------------------------------------------------------
      do 90 n = 1, npop
        ipar1(n) = 0
        ipar2(n) = 0
60      if (ifile .eq. 0) then
          do 70 i = 0, npts-ips-1
            iseed = iseed + 2
            ir = int(npool*rand())+1
            if (ipsp(ir) .eq. 1) goto 60
            do 70 j = 1, ndv
              pop(n,i*ndv+j) = xpool((j-1)*npool+ir)
70        continue
        else
          do 75 i = 0, npts-1
            do 75 j = 1, ndv
              iseed = iseed + 2
              ir = int(idim(j)*rand())
              pop(n,i*ndv+j) = xpool((ir-1)*ndv+j)
75        continue
        end if
c       CHECK AGAINST DUPLICATE POINTS
        do 90 i = 1, npts-ips-1
          do 90 j = i+1, npts-ips
            cc = 0
            do 80 k = 1, ndv
              if (pop(n,(i-1)*ndv+k) .eq. (pop(n,(j-1)*ndv+k)))
                cc = cc + 1
80          continue
            if (cc .eq. ndv) goto 60
90    continue

c     ------------------------------
c     Loop over Number of Generations
c     ------------------------------
      do 500 igen = 1, ngen
c       ------------------------------------------------
c       Evaluate Objective Function for Entire Population
c       i.e. compute det|xTx|
c       ------------------------------------------------
        do 150 n = 1, npop

          do 120 i = 1, npts-ips
            do 100 j = 1, ndv
              xtemp(j) = pop(n,(i-1)*ndv+j)
100         continue
            call getxl(ifit, ncoef, ndv, xtemp, xline)
            do 110 j = 1, ncoef
              x(i+ips,j) = xline(j)
110         continue
```

```
120       continue
          do 130 i = 1, ips
            do 130 j = 1, ncoef
              x(i,j) = xp(i,j)
130       continue

c         ------------------------------------
c         Compute Det|xTx| via QR Factorization
c         Det|xTx| = Det|R|^2
c         ------------------------------------
          call dgeqr2(npts, ncoef, x, npts, w(pwork), w(ptau), info)
          d(n) = 1.0d0
          do 140 i = 1, ncoef
            d(n) = d(n)*x(i,i)*x(i,i)
140       continue
150     continue

c       ------------------------------------
c       Compute Rank and Average Determinant
c       ------------------------------------
        call sort(npop, d, iord)
        aved = 0.0
        do 160 j = 1, npop
          aved = aved + d(j)
160     continue
        aved = aved/dreal(npop)

c       ------------------------
c       Print Generation History
c       ------------------------
        if ((igen .eq. 1) .or. (mod(dreal(igen),10) .eq. 0) ) then
          write(3,915) igen, d(iord(1)), aved
        end if
        if (iprint .eq. 1) then
          write(*,915) igen, d(iord(1)), aved
        else if (iprint .eq. 2) then
          write(*,*)
          write(*,*)
          write(*,903) 'GENERATION NUMBER: ', igen
          write(*,904) 'Child', 'Genes    '
          do 170 i = 1, ndv*(npts-ips)-1
            write(*,905) '       '
170       continue
          write(*,906) 'Par. 1', 'Par. 2', 'Rank', 'Det'
          do 190 i = 1, npop
            do 180 j = 1, npop
              if (iord(j) .eq. i) then
                irank = j
              end if
180         continue
            write(*,907) i
            write(*,908) (pop(i,j), j = 1, ndv*(npts-ips))
            write(*,909) ipar1(i), ipar2(i), irank, d(i)
190       continue
          write(*,910) 'MAXIMUM DETERMINANT: ', d(iord(1))
```

```
          write(*,910) 'AVERAGE DETERMINANT: ', aved
        end if

c       ------------------------------------------------
c       Making Children Designs; Loop Over Population - 1
c       ------------------------------------------------
        do 400 k = 1, npop-1
c         -------------------
c         Selecting Parent #1
c         -------------------
210       iseed = iseed + 2
          xr = rand()
          do 220 i = 2, npop
            if (xr .lt. p(i)) then
              ipar1(k+1) = iord(i-1)
              goto 230
            end if
220       continue
          ipar1(k+1) = iord(npop)

c         ------------------------
c         Selecting Parent #2
c         Parent 2 .ne. Parent 1
c         ------------------------
230       iseed = iseed + 2
          xr = rand()
          do 240 i = 2, npop
            if (xr .lt. p(i)) then
              ipar2(k+1) = iord(i-1)
              if (ipar2(k+1) .eq. ipar1(k+1)) goto 230
              goto 250
            end if
240       continue
          ipar2(k+1) = iord(npop)
          if (ipar2(k+1) .eq. ipar1(k+1)) goto 230

c         ------------------------------------------------------
c         Creating Children (at least 1 gene from both parents)
c         ------------------------------------------------------
250       if (ispace .eq. 0) then
c           --------------------
c           Cubodial Design Space
c           --------------------
            iseed = iseed + 2
            ir = int((ndv*(npts-ips)-1)*rand()) + 1

            do 260 i = 1, ir
              kid(k,i) = pop(ipar1(k+1),i)
260         continue

            do 270 i = ndv*(npts-ips), ir+1, -1
              kid(k,i) = pop(ipar2(k+1),i)
270         continue

c           MUTATIONS
```

```
              do 280 i = 0, npts-ips-1
                do 280 j = 1, ndv
275               iseed = iseed + 2
                  xr = rand()
                  if (xr .lt. xmut) then
                    iseed = iseed + 2
                    if (ifile .eq. 0) then
                      ir = int(npool*rand()) + 1
                      if (ipsp(ir) .eq. 1) goto 275
                      kid(k,i*ndv+j) = xpool((j-1)*npool+ir)
                    else
                      ir = int(idim(j)*rand(j))+1
                      kid(k,i*ndv+j) = xpool((ir-1)*ndv+j)
                    end if
                  end if
280             continue

c             CHECK AGAIN DUPLICATE POINTS
              do 295 i = 1, npts-ips-1
                do 295 j = i+1, npts-ips
                  cc = 0
                  do 290 kk = 1, ndv
                    if (kid(k,(i-1)*ndv+kk) .eq.
     .                    (kid(k,(j-1)*ndv+kk)))
     .                cc = cc + 1
290               continue
                  if (cc .eq. ndv) goto 210
295             continue
            else
c             --------------------
c             General Design Space
c             --------------------
              iseed = iseed + 2
              ir = ndv*(int((npts-ips-1)*rand()) + 1)

              do 300 i = 1, ir
                kid(k,i) = pop(ipar1(k+1),i)
300           continue

              do 310 i = ndv*(npts-ips), ir+1, -1
                kid(k,i) = pop(ipar2(k+1),i)
310           continue

c             MUTATIONS
              do 380 i = 0, npts-ips-1
320             iseed = iseed + 2
                xr = rand()
                if (xr .lt. xmut) then
                  iseed = iseed + 2
                  ir = int(npool*rand()) + 1
                  if (ipsp(ir) .eq. 1) goto 320
                  do 340 j = 1, ndv
                    kid(k,i*ndv+j) = xpool((j-1)*npool+ir)
340               continue
                end if
```

```
380         continue

c           CHECK AGAIN DUPLICATE POINTS
            do 395 i = 1, npts-ips-1
              do 395 j = i+1, npts-ips
                cc = 0
                do 390 kk = 1, ndv
                  if (kid(k,(i-1)*ndv+kk) .eq.
     .                 (kid(k,(j-1)*ndv+kk)))
     .                 cc = cc + 1
390             continue
                if (cc .eq. ndv) goto 210
395         continue

          end if
400     continue

c       ---------------------------------------------------
c       Update Generation (keep best parent, replace parents
c       with children
c       ---------------------------------------------------
        do 410 i = 1, ndv*(npts-ips)
          pop(1,i) = pop(iord(1),i)
410     continue
        do 420 i = 2,  npop
          do 420 j = 1, ndv*(npts-ips)
            pop(i,j) = kid(i-1,j)
420     continue

500   continue

c     ---------------------------------------------------
c     Evaluate Objective Function for the Final Generation
c     ---------------------------------------------------
      do 600 n = 1, npop

        do 530 i = 1, ips
          do 530 j = 1, ncoef
            x(i,j) = xp(i,j)
530     continue
        do 550 i = 1, npts-ips
          do 570 j = 1, ndv
            xtemp(j) = pop(n,(i-1)*ndv+j)
570       continue
          call getxl(ifit, ncoef, ndv, xtemp, xline)
          do 560 j = 1, ncoef
            x(i+ips,j) = xline(j)
560       continue
550     continue

c       -------------------
c       Compute Determinant
c       -------------------
        call dgeqr2(npts, ncoef, x, npts, w(pwork), w(ptau), info)
        d(n) = 1.0d0
```

```
        do 590 i = 1, ncoef
          d(n) = d(n)*x(i,i)*x(i,i)
590     continue

600 continue

c     -------------------------------
c     Compute Rank of Final Generation
c     -------------------------------
      call sort(npop, d, iord)

c     --------------
c     Output Results
c     --------------
      write(*,*)
      write(*,911) 'THERE ARE ', xnop, ' COMBINATIONS'
      write(*,911) 'POOL OF ', real(npool), ' POINTS'
      write(*,912) 'POPULATION SIZE:    ', npop
      write(*,912) 'NO. OF GENERATIONS: ', ngen
      write(*,913) 'THE MAXIMUM DETERMINANT IS ', d(iord(1)),
     .             ' USING POINTS'

      if (ips .ne. 0) then
        do 700 i = 1, npool
          if (ipsp(i) .eq. 1) then
            write(*,914) (xpool((j-1)*npool+i), j = 1, ndv)
            write(7,914) (xpool((j-1)*npool+i), j = 1, ndv)
            write(7,*)
            write(*,*)
          end if
700     continue
      end if

      do 710 i = 1, npts-ips
        do 705 j = 1, ndv
          denom = pop(iord(1),(i-1)*ndv+j)
          write(*,914) denom
          write(7,914) denom
705     continue
        write(*,*)
        write(7,*)
710   continue
      write(*,*)
      close(3)
      close(7)

903   format(a,5x,i3)
904   format(a5,2x,a8,$)
905   format(a,$)
906   format(1x,a6,1x,a6,2x,a4,3x,a3)
907   format(i5,$)
908   format(1x,f6.2,$)
909   format(1x,i6,1x,i6,4x,i4,1x,1pe10.2)
910   format(a,1pe10.2)
911   format (/a,1pe10.4,a)
```

```
912   format (a,i5)
913   format (a,1pe12.5,a/)
914   format (2x,f10.5,$)
915   format (2x, i6, 2x, 2(1pe10.4,2x))

      end

c-----------------------------------------------------------------------
      subroutine evals(npool, npts, ips, x)

c     Computes the number of evaluations of xTx that would have to be
c     made if every combination were tried to find the D-optimal points

c     -------------------
c     Variable Definition
c     -------------------
c     ---------------
c     Input variables
c     ---------------
c     npool            number of points to choose from
c     npts             number of points to choose
c     ips              number if preselected points

c     ----------------
c     Output variables
c     ----------------
c     x                number of combinations of (npts-ips) from npool
c
c     -----------------
c     Working variables
c     -----------------
c     i                counter
c-----------------------------------------------------------------------
c     --------------
c     Variable Block
c     --------------
      integer npool, npts, ips, i
      real*8  x

c     ------------------------------------------------
c     Compute (npool)!/((npool-(npts-ips))!(npts-ips)!)
c     ------------------------------------------------
      x = 1.0d0
      do 5 i = 2, npts-ips
        x = x*dreal(i)
5     continue

      x = 1.d0/x
      do 10 i = npool-npts-2*ips+1, npool-ips
        x = x*dreal(i)
10    continue

      return
      end
```

```
c-----------------------------------------------------------------------
      subroutine getxl(ifit, ncoef, idim, x, xline)

c     This subroutine computes a row of the least-squares matrix given
c     one of the data points
c
c     -------------------
c     Variable Definition
c     -------------------
c     ---------------
c     Input Variables
c     ---------------
c     ifit              Type of function to describe the response surface
c     ncoef             No. of coefficients in polynomial used to describe
c                         the response surface
c     idim              dimension of design space
c     x                 Array of dimension idim; contains data point of
c                         row of least-squares matrix
c
c     ----------------
c     Output Variables
c     ----------------
c     xline             Array of dimension ncoef; contains value of row
c                         for least squares matrix
c     -----------------
c     Working Variables
c     -----------------
c     i, j, k           counters
c     cc                counter identifying coefficient
c-----------------------------------------------------------------------
c     --------------
c     Variable Block
c     --------------
      integer i, j, k, cc, ifit, ncoef, idim
      real*8 x(idim), xline(ncoef)

c     ----------------------------------------
c     Construct row of matrix according to ifit
c     ----------------------------------------
      if (ifit .le. 1) then
        cc = 0
        if (ifit .eq. 1) then
c         QUADRATIC TERMS
          do 660 i = 1, idim
            do 660 j = i, idim
              cc = cc + 1
              xline(cc) = x(i) * x(j)
660       continue
        end if
c       LINEAR TERMS
        do 670 i = 1, idim
          cc = cc + 1
          xline(cc) = x(i)
670     continue
c       CONSTANT
```

```
        cc = cc + 1
        xline(cc) = 1.0d0
      else if (ifit .eq. 2) then
c       FULL LINEAR
        if (idim .eq. 2) then
          cc = 0
          do 674 i = 0, 1
            do 674 j = 0, 1
              cc = cc + 1
              xline(cc) = x(1)**i * x(2)**j
674       continue
        else if (idim .eq. 3) then
          cc = 0
          do 684 i = 0, 1
            do 684 j = 0, 1
              do 684 k = 0, 1
                cc = cc + 1
                xline(cc) = x(1)**i * x(2)**j * x(3)**k
684       continue
        end if
      else if (ifit .eq. 3) then
c       FULL QUADRATIC
        if (idim .eq. 2) then
          cc = 0
          do 675 i = 0, 2
            do 675 j = 0, 2
              cc = cc + 1
              xline(cc) = x(1)**i * x(2)**j
675       continue
        else if (idim .eq. 3) then
          cc = 0
          do 685 i = 0, 2
            do 685 j = 0, 2
              do 685 k = 0, 2
                cc = cc + 1
                xline(cc) = x(1)**i * x(2)**j * x(3)**k
685       continue
        end if
      else if (ifit .eq. 4) then
        xline(1) = x(1)*x(2)*x(2)
        xline(2) = x(1)*x(2)
        xline(3) = x(1)
        xline(4) = x(2)*x(2)
        xline(5) = x(2)
        xline(6) = 1.d0
      else if (ifit .eq. 5) then
        xline(1) = x(1)*x(1)*x(2)
        xline(2) = x(1)*x(1)
        xline(3) = x(1)*x(2)
        xline(4) = x(1)
        xline(5) = x(2)
        xline(6) = 1.d0
      end if
      return
      end
```

APPENDIX
C

ErICA USER'S GUIDE

ErICA or EuleR Inviscid Code for Aerodynamics is a two-dimensional Euler solver for perfect gas flows. ErICA is written in FORTRAN 77 with a C front end for dynamic memory allocation. It has been validated with a number of flows from shock reflection problems to flow over transonic airfoils. The code features a variety of boundary conditions to solve many, diverse types of flows. For example, Dadone/Grossman tangency conditions assist in modeling flows around curved surfaces. Far field boundary conditions are available to solve the small disturbance potential equation for accurate estimations of the lift and drag of an airfoil. Numerical stability of the code is achieved through upwind MUSCL differencing of the flux. Riemann inflow/outflow conditions propagate information along characteristic lines and also add numerical stability at the boundaries. Up to third-order spatial accuracy can be achieved for solutions using the interpolating polynomial in the upwind solver. ErICA also features a number of convergence accelerators such as mesh sequencing and multigrid. For flows which do not contain stagnation points, preconditioning is offered as an alternate convergence accelerator. ErICA's output is formatted for the immediate use in TECPLOT^{TM} or PLOT3D^{TM}. Also an unformatted solution file is generated for restart or data messaging.

This appendix is intended to explain how to use the code to run analyses for two-dimensional problems. This includes a description of the set of four data files useful for running the code.

C.1 ErICA's Main Input Deck

ErICA is driven from a series of files which describe the flow problem. The main input deck supplies general information of the problem and how the problem will be solved by ErICA. A file containing the computational grid serves to describe the geometry of the problem. For many problems these files will be enough to run ErICA. If one or more of the boundaries require user specific information, then a third file is required to furnish this information. Also, the code can be "warm-started," through a restart file.

The code is run on UNIX-based systems by typing

$$erica < filename$$

where filename refers to the name of the main input deck. The grid, boundary and restart file names are supplied to ErICA through the main input deck.

The main input deck is designed to allow the user to run of variety of aerodynamic problems at various conditions with minimal effort involved in setting up the problem. The file is segmented into nine sections: reference quantities, free stream and gas constants, initialization, grid data, time integration, boundary condition flags, spatial accuracy, residual smoothing, mesh sequencing, multigrid, and output flags. These sections are listed below with a brief description. All dimensional variables must be entered using the SI system with angles represented in degrees. A sample input deck follows. It is important to follow the sample deck carefully as all the quantities are read with format statements.

Reference Quantities

Calculations for numerically solving the two-dimensional Euler equations are performed using a non-dimensional system using a reference velocity, temperature

and pressure. Choosing these to be numerically equal to one results in dimensional calculations. It is possible to reduce round-off error by choosing the reference conditions so that the non-dimensional flow variables are all of the same order. The non-dimensionalization scheme used preserves the forms of the governing equations as well as the ideal gas equation of state. The format statement for this card is (3(f10.3)).

Free Stream Conditions and Gas Constants

The free stream state is specified using Mach number, angle of attack, temperature and pressure. ErICA uses the gas constant, R, and the ratio of specific heats, g, to relate thermodynamic properties. Here g is assumed constant over all possible ranges of temperature. These constants are available for common gases in several sources, *e.g.* The Handbook of Chemistry and Physics published by CRC Press. For air, these constants are 287 J/(kg k) and 1.4 respectively. The format for this card is (6(f10.3)).

Initialization

ErICA can be initialized to either a free stream state or a general state described through a restart file. The free stream/restart flag should be set to 0 for a start from free stream or 1 for a restart. Following the free stream/restart flag is the name of the restart file (under 20 characters). If no restart file is needed, this space should read 'none'. The format for this card is (i10,20x,a20).

Grid Data

This section supplies ErICA with information on how to interpret the grid. The size of the grid is indicated by the first two numbers. The next two numbers must

be set to zero unless ErICA is to solve the flow around an airfoil using a C-grid. In this specific case these numbers indicate the j-indices of the trailing edge (lower and upper surface). Next the file containing the grid points is specified. The format for this card is (4(i10),a20).

Time Integration

Here, the user specifies six quantities associated with the time marching integration. The first is the maximum number of iterations ErICA will perform. After this amount, integration will stop and ErICA will exit normally, generating restart and output files. The next integer is the CFL flag. If set to 0, the time integration is performed with the constant time step indicated by the next number. If 1 is specified, the integration is performed with the constant CFL defined by the fourth number. If the CFL flag is 0, the CFL is ignored, else the time step is ignored. A steady state solution has been reached when the normalized residual is equal to zero. ErICA will stop time marching when it has reached 0 to within the user specified stopping tolerance given by the next number. With the correct conditions ErICA is capable of reaching a double precision machine zero stopping tolerance, but many times such a tight tolerance is not needed. The final number specified in this section dictates the time integration method. If 0, ErICA will use an implicit approximate factorization method. If between 1 and 4, ErICA will do that many stages of Runge-Kutta explicit iterations. The format for this card is (i10,i10,3(f10.3),i10).

Boundary Conditions

The next set of flags refers to the conditions to be enforced on each boundary of the computational domain. The conditions are specified for each boundary in

the order of $j = 1$, $j = jdim$, $k = 1$, and $k = kdim$. The condition type is set by assigning the corresponding integer to the boundary:

0.)	free stream
1.)	extrapolate all
2.)	fix at specified values
3.)	tangency (Dadone/Grossman)
4.)	subsonic inflow/outflow
5.)	cut-line with tangency (Dadone/Grossman)
6.)	Riemann Invariants
7.)	tangency (symmetry technique)
8.)	cut-line with tangency (symmetry technique)
9.)	airfoil far field (kdim only)
10.)	cut-line

If any or all boundary conditions are set to 2, the name of the data file containing the user specified values must be given (one file for any and all boundaries). If no file is necessary, this space should read 'none.' The format for this card is (4(i10),a20).

Spatial Accuracy

Spatial accuracy is set by choosing values for phi and kappa. With phi = 0, the solution will be first-order throughout. Phi = 1 and kappa = 1/3 yields a globally third-order solution. Limiters can be turned on to remove unwanted oscillations near large gradients by making the solutions locally first-order. The choices for the limiters are

0.)	no limiter
1.)	minmod applied to primitive variables
2.)	minmod applied to characteristic variables
3.)	Van Albada applied to primitive variables

The next flag toggles the entropy modification routine. If set to 0, no entropy modification is done; if set to 1, the entropy near stagnation points and points where the velocity reaches the speed of sound will be slightly modified to avoid standing expansion fans. The format for this card is (f10.3,f10.7,4(i10)).

Residual Smoothing

The residual smoothing flag, when set to 1 will average the residual of a cell to surrounding neighbors weighted according to a parameter epsilon. Convergence for certain problems can improve when the residual is smoothed. The next number is epsilon and should be of the order of 0.5. The format for this card is (i10,f10.5).

Mesh Sequencing

Mesh sequencing is a technique whereby time integration to steady state begins with using coarse grids. Iterations on coarse grids are cheaper to perform and can have significant time savings. ErICA is capable of a three level mesh sequence, provided the grid meets certain specifications. For a two level mesh sequence, all the grid dimensions ($jdim$, $kdim$, TE#1, TE#2) must satisfy the condition

$$mod(grid\ dimension + 1, 2) = 0.$$

Three levels are permitted provided all grid dimensions satisfy

$$mod(grid\ dimension + 3, 4) = 0.$$

The first number in this section of the input deck specifies the levels of mesh sequencing desired. If neither of the above conditions can be met, this number must be zero, otherwise it is set to the number of levels minus one. The next two numbers, $ims(1)$ and $ims(2)$, indicate how many iterations are performed on the lower and medium levels. If one level is performed, $ims(1)$ must be set to 0. If mesh sequencing is not used, these numbers are ignored. The format for this card is (4(i10)).

Multigrid

Multigrid is an convergence acceleration technique whereby several grids of varying density are used in a cycle. For more details on multigrid schemes see Ref. 1. Up to three levels of grids are available provided the same conditions for mesh sequencing are met. If two level multigrid is used, a cycle consists of fine grid iterations followed by medium grid iterations followed by fine grid iterations. If three level multigrid is used, a cycle consists of fine grid iterations, followed by medium grid iterations, followed by coarse grid iterations followed by fine grid iterations. Each cycle makes up one global time iteration. The first number in this section is the number of multigrid levels minus one. The next four numbers, indicate how many iterations on each level will be performed during the multigrid cycle. These numbers are ignored if the multigrid option is not chosen. The format for this card is (5(i10)).

Output

The output section is broken into three parts. The first part contains three flags. The first flag, if 0 will print the solution at cell centers, or if 1 will print the solution at the grid points. The next flag formats the output file for TECPLOT^{TM}

(0) or PLOT3D^{TM} (1) applications. The next flag if 0 suppresses all output to the screen. If 1, then a residual history is written to the screen. The format for this card is (i10,10x,i10,10x,i10).

The residual is sent to the screen or a file at a frequency corresponding to the first number in this section. For example, if the frequency is 10, the residual history is updated every 10 iterations. The next piece of information is the name of the file to contain the residual history. The format for this card is (i10,20x,a20).

Output and restart files are generated at a frequency corresponding the first number in this section. The next two pieces of data are the name of the output and restart file to be generated. The format for this card is (i10,20x,2(a20)).

The main input deck is shown below

ErICA Deck
written by Bob Narducci - June, 1994
Version 1.0

Reference Quantities

Vref	Tref	Pref	(SI units)
374.16	348.43	1.e+5	

Free Stream Conditions/Gas Quantities

Mach	AOA	Temp	Pressure	Gamma	Gas Constant
0.75	0.00	348.43	1.e+5	1.4	287.0

Initialization

Freestream/Restart file	Restart Filename
0	des2.rst

Grid Data

jdim	kdim	T.E.#1	T.E.#2	Filename
201	53	41	161	../AIRFOIL/des2.grd

Time Integration

maxiter	CFL Flag	delta t	CFL	Stop Tol	imp/R-K stages
2	1	5.0d-5	30.0	1.0d-6	0

Boundary Condition Flags

j=0	j = jdim	k = 0	k = kdim	Filename
9	9	5	9	none

Spatial Accuracy					
phi	kappa	limiter	Flux	Ent Fix	Precond.
1.0	.33333333	3	0	0	0

Residual Smoothing	
On/Off	epsilon
0	0.5

Mesh Sequencing			
levels	CG Iter.	MG Iter	CG MG Iter
2	200	100	0

Multigrid				
levels	FG Cycl	MG Cycl	CG Cycl	FG Cycl
0	2	3	10	1

Output			
Cell Ctr/Grid Pts	Tecplot/Plot3d	Output to Screen	
1	0	1	
Freq. Resid sent to file	Residual Filename		
1	des2.res2		
Freq. output files are Gen.	Solution Filename	Restart Filename	
20	des2.out	des2.rst2	

C.2 Grid File

The grid file consists of two columns which make up the (x,y) pairs of grid points. The file is read using the following FORTRAN lines

```
      do 300 j = 1, jdim
         do 200 k = 1, kdim
            read(12,*) x(j,k), y(j,k)
200      continue
300   continue
```

C.3 Boundary Condition File

The boundary condition file allows point by point specification of the density, x- and y-components of the velocity, and pressure along the edges of the flow domain.. The flow state must be specified for every cell of each boundary flagged with a 2 in the boundary condition section of the main input deck. ErICA is prepared to read in the data starting with the $j = 1$ boundary (if necessary), then the $j = jdim$ boundary (if necessary), followed by the $k = 1$ boundary (if necessary), and finally the $k = kdim.$ (if necessary). The file is read using the following FORTRAN line

```
read(12,*) qbc1(i,1), qbc1(i,2), qbc1(i,3), qbc1(i,4)
```

C.4 Restart File

The restart file allows ErICA to be warm started. The restart file generated by ErICA can be used without modification to continue iterations to steady state. The file is read using the following FORTRAN lines

```
      read(*) nlast, norm1
      do 100 i = 1, ivol
           read(*) (q(i,n), n = 1, 4)
100   continue
```

The first line of the file contains the previous iteration number and the value of the normalized residual using free stream conditions. These values are not critical in obtaining a solution. If *nlast* and *norm*1 are not available, they should be set to one.

VITA

The author was born in Worcester, Massachusetts on April 22, 1968. In 1976, his family moved to the suburbs of Philadelphia, where he became an avid Phillies fan. He attended Haverford High School specializing in Science and Mathematics. In Chemistry class, on January 28, 1986, moments after the Challenger explosion, the Robert decided to pursue a career in aerospace engineering. After completing his secondary education in 1986, Robert attend Virginia Tech where he majored in Aerospace and Ocean Engineering. While working on his Bachelors of Science degree, he participated in the Cooperative Education program, working as a project engineer at the Naval Air Test Center in Patuxent River, Maryland. After graduating in 1991 with dual degrees in Aerospace Engineering and Ocean Engineering, Robert entered the graduate school at Virginia Tech. On July 18, 1993, he married Erika Markussen. Robert is currently employed with McDonnell-Douglas Aerospace in Long Beach, California.