Airfoil Design by an All-At-Once Method

Ajit Shenoy, Matthias Heinkenschloss, and Eugene M. Cliff

CRPC-TR97703-S
November 1997

Center for Research on Parallel Computation
Rice University
6100 South Main Street
CRPC - MS 41
Houston, TX 77005

Submitted in November 1997

Report Documentation Page

Form Approved
OMB No. 0704-0188

1. REPORT DATE	2. REPORT TYPE	3. DATES COVERED
05 NOV 1997		**00-00-1997 to 00-00-1997**

4. TITLE AND SUBTITLE	5a. CONTRACT NUMBER
Airfoil Design by an All-At-Once Method	5b. GRANT NUMBER
	5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S)	5d. PROJECT NUMBER
	5e. TASK NUMBER
	5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) **Rice University,Department of Computational and Applied Mathematics,Houston,TX,77005-1892**	8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSOR/MONITOR'S ACRONYM(S)
	11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution unlimited

13. SUPPLEMENTARY NOTES

14. ABSTRACT

The all?at?once approach is implemented to solve an optimum airfoil design problem. The airfoil design problem is formulated as a constrained optimization problem in which flow variables and design variables are viewed as independent and the coupling steady state Euler equation is included as a constraint, along with geometry and other constraints. In this formulation, the optimizer computes a sequence of points which tend toward feasiblility and optimality at the same time (all?at?once). This decoupling of variables typically makes the problem less nonlinear and can lead to more efficient solutions. In this paper an existing optimization algorithm is combined with an existing flow code. The problem formulation, its discretization, and the underlying solvers are described. Implementation issues are presented and numerical results are given which indicate that the cost of solving the design problem is approximately six times the cost of solving a single analysis problem.

15. SUBJECT TERMS

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT **unclassified**	b. ABSTRACT **unclassified**	c. THIS PAGE **unclassified**	**Same as Report (SAR)**	**34**	

Airfoil Design by an All–At–Once Method *

Ajit Shenoy † Matthias Heinkenschloss ‡ Eugene M. Cliff §

Abstract

The all–at–once approach is implemented to solve an optimum airfoil design problem. The airfoil design problem is formulated as a constrained optimization problem in which flow variables and design variables are viewed as independent and the coupling steady state Euler equation is included as a constraint, along with geometry and other constraints. In this formulation, the optimizer computes a sequence of points which tend toward feasiblility and optimality at the same time (all–at–once). This decoupling of variables typically makes the problem less nonlinear and can lead to more efficient solutions. In this paper an existing optimization algorithm is combined with an existing flow code. The problem formulation, its discretization, and the underlying solvers are described. Implementation issues are presented and numerical results are given which indicate that the cost of solving the design problem is approximately six times the cost of solving a single analysis problem.

Key words Airfoil design, optimization, computational fluid dynamics, Euler equations, nonlinear programming, optimal design.

1 Introduction

Optimum airfoil design is an active area of research. See, *e.g.*, the recent papers [2], [4], [18], [19], [20], [22], [23], [26], [27], [33], [34]. Abstractly, the optimum airfoil design problem can be formulated as a constrained optimization problem, and many techniques have been applied to its solution. Most of the recent approaches, in fact all of the references above, combine optimization and optimal control techniques with computational fluid dynamics. Several issues have to be dealt

*This version was generated November 5, 1997.

†Interdisciplinary Center for Applied Mathematics and Aerospace Engg. Dept., Virginia Polytechnic Institute and State University, Blacksburg, Virginia 24061–0531. This author was supported by AFOSR under Grants F49620–93–1–0280 and F49620–96–1–0329.

‡Department of Computational and Applied Mathematics, Rice University, Houston, Texas 77005–1892, USA, E-Mail: heinken@rice.edu. This author was supported by NSF under Grant DMS–9403699, by DoE under Grant DE-FG03-95ER25257, AFOSR under Grant F49620–96–1–0329.

§Interdisciplinary Center for Applied Mathematics and Aerospace Engg. Dept., Virginia Polytechnic Institute and State University, Blacksburg, Virginia 24061–0531, USA, E-Mail: cliff@icam.vt.edu. This author was supported by AFOSR under Grants F49620–93–1–0280 and F49620–96–1–0329.

with when one follows this path. Among these are the formulation of the optimization problem, the discretization of the infinite dimensional airfoil design problem, and the differentiation of objective function and constraints. For the solution of the airfoil design problem all issues have to be dealt with simultaneously. Great care must be taken to avoid, or at least to control inconsistencies and to develop a robust and efficient solution method. While these techniques have already been successfully applied to airfoil design problems, further investigations are needed to improve efficiency and robustness of the solution techniques and to increase the set of (airfoil) design problems to which these techniques can be applied.

In this paper we investigate the applicability of the all–at–once formulation of the optimization problem to solve an airfoil design problem. In most papers on airfoil design, the flow variables q are viewed as functions of the design parameters w. This function $q(w)$ is implicitly defined by the governing equations $R(q, w) = 0$, in our case the steady state 2-D Euler equations. The optimization formulation describing the airfoil design problem is then posed in the design variables w. This is called the black-box approach. The Euler equations are not visible to the optimizer, but hidden by eliminating the flow variables, *i.e.*, by expressing the flow variables q as functions of the designs variables w. An alternative to this approach is the all–at–once formulation in which one views flow variables q and design variables w as independent variables in the optimization problem. The Euler equations coupling these two are included into the optimization formulation as a constraint along with other constraints such as geometric constraints, drag constraints, etc. The optimizer is now responsible for computing a point which is feasible and optimal at the same time, *i.e.*, move towards feasibility and optimality at once, rather than moving along the manifold of feasible points towards optimality. Comparisons between these two approaches on other problems have shown that the all–at–once approach can be substantially faster. The reason is that viewing q and w as independent variables, allows the optimizer to violate the Euler equations during the iterations. These are only required to be satisfied at the solution. This makes the optimization problem less nonlinear and often results in fewer iterations. For example, Iollo *et al.* [18] report that their pseudo-time stepping implementation of the all–at–once approach requires only three to four times as many iterations to solve the design problem as compared to the effort required for the solution of a single analysis problem. Our results indicate a factor of five or six. However, our optimization approach is different and our problem includes geometric constraints. It is also important to note that an optimizer implementing the all–at–once approach requires roughly the same problem information as an optimizer applied to the black-box approach, except that the all–at–once approach does not require solutions to the nonlinear flow equations. We give a more detailed presentation of the relations in the next section.

Rather than formulating the airfoil design problem, its discretization, and a solution algorithm and then implement all components from scratch, we decided to build upon existing codes. In our implementation of the all–at–once method for our airfoil design problem we combine the optimizer, TRICE, with the flow code, ErICA. This imposes certain limits on the choice of problem formulation and discretization, but we believe this to be a realistic approach. As we have indicated above, various issues have to be addressed when solving the airfoil design problem. We focus on the optimization formulation. Our airfoil parameterization is obtained by choosing a set of basis airfoils and computing an optimized airfoil as a linear combination of those. Moreover,

grid generation and discretization of the Euler equation was done to limit difficulties arising from nonsmoothness and inconsistencies. Further gains in efficiency and accuracy can be achieved by using more refined discretization techniques and improving the coupling of the flow solver with the optimizer. This was beyond the scope of this study and is planned for future investigations.

This paper is organized as follows: In Section 2 we discuss optimization formulations and their relations. This section also provides further motivation for the all–at–once approach and reviews some existing optimization approaches. The governing equations and the flow code ErICA are discussed in Section 3. The design problem is formulated in Section 4 and Section 5 contains a description of the optimizer TRICE. Section 6 contains some implementation issues that have to be resolved when combining an optimizer with a flow code for our situation. Section 7 presents some numerical results and contains a discussion of our numerical experiments and open issues.

2 Optimization Problem

There are several ways to cast the design problem outlined in the introduction into an optimization problem. Two formulations will be discussed in this section. The main purpose of this section is to provide a background for the discussion of our approach to the airfoil design problem and for a comparison with other approaches in the literature. In this section we proceed as follows: First, we present the two formulations and their relation in an abstract framework. Then we discuss the applicability to the airfoil design problem.

The first formulation of the airfoil design problem is given by

$$\begin{aligned} \min \quad & J(q,w), && (2.1)\\ \text{s.t.} \quad & R(q,w)=0, && (2.2)\\ & G(q,w)\le 0. && (2.3) \end{aligned}$$

Here q represent the flow variables and w are the design parameters. The constraint function R represents the Euler equations. The inequality constraints (2.3) represent geometric constraints, drag constraints and the like. The special case in which G does not depend on the flow parameters q deserves attention. In this section we assume that the functions $J : \mathbb{R}^{n_q} \times \mathbb{R}^{n_w} \to \mathbb{R}$, $R : \mathbb{R}^{n_q} \times \mathbb{R}^{n_w} \to \mathbb{R}^{n_q}$, and $G : \mathbb{R}^{n_q} \times \mathbb{R}^{n_w} \to \mathbb{R}^{n_g}$ are twice continuously differentiable at the points under consideration. However, we note that for the formulation and execution of the optimization algorithm applied to our airfoil design problem, we only need first derivatives. For R, G we denote differentiation with respect to a variable by using the variable as a subscript, *e.g.*, $R_q(q,w)$ denotes the partial Jacobian of R with respect to q. In addition to the differentiability assumption, we make the assumption that $R_q(q,w)$ is invertible at all points (q,w) under consideration.

Under the assumption of the implicit function theorem (see, *e.g.*, [24]), the constraint (2.2) locally defines a function $q : \mathbb{R}^{n_w} \to \mathbb{R}^{n_q}$ as the solution of

$$R(q(w),w)=0. \tag{2.4}$$

If the equation (2.2) has a unique solution $q(w)$ for all $w \in \mathbb{R}^{n_w}$ under consideration (typically, (2.3) represents an explicit restriction of the design space and therefore not the whole $\mathbb{R}^{n_w}$ is

relevant), then we can eliminate the flow variables q and formulate (2.1)–(2.3) in the following reduced form:

$$\min \quad \hat{J}(w) = J(q(w), w), \tag{2.5}$$

$$\text{s.t.} \quad \hat{G}(w) = G(q(w), w) \leq 0. \tag{2.6}$$

The optimization formulation (2.1)–(2.3) corresponds to the all–at–once (AAO) approach [7], [11] (also called the simultaneous analysis and design (SAND) approach [3], [26]). The optimization formulation (2.5), (2.6) corresponds to the black–box approach [11] (also called the nested analysis and design (NAND) [3], [26]) and it corresponds to the multidiscipline feasible and individual discipline feasible approach in [7].[1]

In the following, we present optimality conditions for (2.1)–(2.3) and we discuss the relation between these two problems. These results are known and can be found in a similar form, *e.g.*, in [9], [13]. Let

$$L(q, w, \lambda, \mu) = J(q, w) + \lambda^T R(q, w) + \mu^T G(q, w) \tag{2.7}$$

be the Lagrangian corresponding to (2.1)–(2.3). If a constraint qualification is met, then for an optimal point (q, w) of (2.1)–(2.3) there exist λ, μ such that

$$\begin{aligned} \nabla_q J(q, w) + R_q(q, w)^T \lambda + G_q(q, w)^T \mu &= 0, \\ \nabla_w J(q, w) + R_w(q, w)^T \lambda + G_w(q, w)^T \mu &= 0, \\ R(q, w) &= 0, \\ G(q, w) &\leq 0, \\ \mu &\geq 0, \\ G(q, w)^T \mu &= 0. \end{aligned} \tag{2.8}$$

If G does not depend on q, then the first equation in (2.8) reduces to

$$\nabla_q J(q, w) + R_q(q, w)^T \lambda = 0. \tag{2.9}$$

Equation (2.9) is called the *adjoint equation* and, if G does not depend on q, defines the Lagrange multiplier (or the co–state) λ. With λ given by (2.9), the term $\nabla_w J(q, w) + R_w(q, w)^T \lambda$ of the second equation in (2.8) is called the *reduced gradient*.

A commonly used constraint qualification is the linear independent constraint qualification (LICQ): Let $G^I(q, w)$ denote the vector of functions of $G(q, w)$ which are active at (q, w). Then LICQ is satisfied if the gradient of the component functions in $R(q, w)$ and $G^I(q, w)$ are linearly independent. If G does not depend on q and if the rows of $G^I(w)$ are linear independent, which is, *e.g.*, the case if $G(w) = \pm w$, then our assumption that $R_q(q, w)$ is invertible implies that LICQ is satisfied [9], [13].

The second order necessary [sufficient] optimality conditions are given by (2.8) and

$$\begin{pmatrix} s_q \\ s_w \end{pmatrix}^T H(q, w, \lambda, \mu) \begin{pmatrix} s_q \\ s_w \end{pmatrix} \underset{[>]}{\geq} 0 \tag{2.10}$$

[1] Since we only have one discipline $R(q, w) = 0$ there is no distinction between the multidiscipline feasible and individual discipline feasible formulation in [7].

for all s_q, s_w satisfying

$$R_q(q,w)s_q + R_w(q,w)s_w = 0, \tag{2.11}$$
$$G_q(q,w)s_q + G_w(q,w)s_w = 0. \tag{2.12}$$

Unless noted otherwise, $G(q,w)$ would usually refer to the set of active constraints, $G^I(q,w)$. Here $H(q,w,\lambda,\mu)$ denotes the Hessian of the Lagrangian

$$H(q,w,\lambda,\mu) = \nabla^2_{(q,w)} L(q,w,\lambda,\mu).$$

Points satisfying the homogeneous state equation (2.11) can be characterized by $\binom{s_q}{s_w} = T(q,w)s_w$, where

$$T(q,w) = \begin{pmatrix} -R_q(q,w)^{-1}R_w(q,w) \\ I \end{pmatrix}. \tag{2.13}$$

With this, (2.10), (2.11), (2.12) can be rewritten as

$$s_w T(q,w)^T H(q,w,\lambda,\mu) T(q,w) s_w \geq [>]\, 0 \tag{2.14}$$

for all s_w satisfying

$$[-G_q(q,w)R_q(q,w)^{-1}R_w(q,w) + G_w(q,w)]s_w = 0. \tag{2.15}$$

The matrix $T(q,w)^T H(q,w,\lambda,\mu)T(q,w)$ is called the *reduced Hessian*. The term on the left hand side of (2.15) can either be computed by calculating the sensitivities $R_q(q,w)^{-1}R_w(q,w)$ or using an adjoint approach. If we define

$$\Lambda = -R_q(q(w),w)^{-T} G_q(q(w),w)^T, \tag{2.16}$$

then the left hand side of (2.15) can be written in the form $\Lambda^T R_w(q(w),w) + G_w(q(w),w)$. In particular if n_g is smaller than n_w the adjoint equation based approach seems more attractive than the sensitivity equation approach.

It is known, see, *e.g.*, [9], [13], that derivatives for the reduced problem (2.5), (2.6) are related to the reduced quantities of the problem (2.1)–(2.3). For example, the gradient $\nabla\hat{J}(w)$ of the reduced problem is equal to the reduced gradient $\nabla_w J(q,w) + R_w(q,w)^T\lambda$, with λ given by (2.9), at $q = q(w)$. Moreover, the Hessian $\widehat{H}(w,\mu) = \nabla^2_w \hat{L}(w,\mu)$ of the Lagrangian $\hat{L}(w,\mu) = \hat{J}(w) + \hat{G}(w)^T\mu$ of the reduced problem is equal to the reduced Hessian $T(q,w)^T H(q,w,\lambda,\mu)T(q,w)$. Finally, the Jacobian $\widehat{G}_w(w)$ is equal to the matrix on the left hand side of (2.15) at $q = q(w)$.

Comparisons between the all–at–once (SAND) approach and the black–box (NAND, discipline feasible) approach can be found in, *e.g.*, [7], [11], [26]. The all–at–once approach decouples state and design variables. An optimizer for (2.1)–(2.3) can use this decoupling and is allowed to violate constraints during the iteration. This can result in substantial gains in performance. Significant reductions in solution times for problems related to the one considered in this paper are reported in [11], [12]. However, the optimizer must achieve feasibility and optimality at the same time. This requires carefully designed optimization codes to maintain robustness. In the

computational experiments in [11] the (particular) implementation of the black–box formulation always performed more robustly than the implementation of the all–at–once approach for a one–dimensional duct design problem. Concerning the applicability of the all–at–once approach and the black–box approach, it should be mentioned that most of the quantities needed to implement the all–at–once approach also have to be provided for an implementation of the black–box approach. This is indicated by the relations between derivatives for the reduced problem (2.5), (2.6) and the reduced quantities of the problem (2.1)–(2.3) summarized above.

In the context of airfoil design problems both formulations (2.1)–(2.3) and (2.5), (2.6) have been used, however, currently the black–box formulation (2.5), (2.6) seems to be dominant [19], [20], [23], [27], [33]. For airfoil design problems the all–at–once formulation (2.1)–(2.3) is considered in [18], [34]. In both cases only the equality constrained problem (2.1), (2.2) is considered. The optimization methods are derived from the optimality system for (2.1), (2.2). In [18] the optimality system is solved by applying a few pseudo–time steps to the Euler equation and the adjoint equation to improve state and co–state estimates and a gradient–like step to update the design variables. The optimization methods in [34] are derived from the application of Newton's method to the optimality system; these are particular versions of sequential programming (SQP) methods.

We use the all–at–once formulation (2.1)–(2.3). For our particular design problem the constraints G are simple constraints on the design variables. The exact problem formulation will be introduced in the subsequent sections. We use an SQP method from the class of methods described in [9], [15] for the solution of the all–at–once formulation. This SQP method uses an interior point strategy to handle the inequality constraints and employs a trust–region strategy for globalization of convergence and to enhance robustness. See also Section 5. If only equality constraints are present, then the Newton based methods in [34] are related to the SQP methods in [9], [15]. Besides the capability of handling inequalities on the designs, other main differences are that the SQP methods in [9], [15] use a trust–region globalization and, in addition to exact second derivatives, provide quasi–Newton approximations to the full and reduced Hessian of the Lagrangian. First and second order convergence results are proven in [9] and the influence of inexact derivatives is analyzed in [14].

For the formulation of the airfoil design problem as an optimization problem, several other issues are of great importance. These are the issues of discretization, differentiability, and unique solvability of state equations and linearized state equations. We give a more detailed description below. For general airfoil design problems comprehensive, rigorous treatments of these issues are still missing. In the case of a one-dimensional duct design problem, which is related to the airfoil design problem, such a comprehensive, rigorous treatment can be found in [6]. It is shown in [6] that an understanding of these issues can be used to improve robustness and efficiency of the optimization code. These improvements are based on the understanding of the problem, of its discretization, and of the optimization method. They are achieved with very little programming effort and almost no additional computing effort per iteration.

The airfoil design problem originally is an infinite dimensional problem. Therefore, the optimization formulation and optimization algorithm have to be combined with a discretization scheme. Various approaches are possible. Two of those are the optimize–then–discretize approach in which the optimization algorithm is formulated in the infinite dimensional setting and then discretization

are applied to the individual steps, and the discretize–then–optimize approach in which one first discretizes the problem and then applies an optimization algorithm to the discretized problem. The processes of discretization and optimization are usually not interchangeable and therefore these two approaches are different. The numerical solution of an infinite dimensional problem requires a careful study of the problem at hand. Several issues have to be kept in mind. In the optimize–then–discretize approach the derivatives after discretization are usually not the derivatives of the discretized functions. Therefore optimization algorithms have to cope with inexact derivative information. See *e.g.*, [5], [33]. The discretize–then–optimize approach often neglects the fact that the infinite dimensional problem structure still influences the finite dimensional problem. If this influence is not incorporated properly, then the optimization problem typically becomes artificially ill–conditioned and one observes a severe degradation in performance and robustness of the optimizer, see [6], [28].

Derivatives of constraint functions and solutions q to the state equations are used in the formulation of optimality conditions and in efficient optimizers. See, for example, gradient computations using sensitivities or adjoint equations. For problems governed by the Euler equations, differentiability in the infinite dimensional context is problematic, due to the presence of shocks. This might be different for the discretized Euler equations. If smoothing procedures (*e.g.*, introduction of artificial viscosity) are applied in discretization schemes for the Euler equations, the resulting finite dimensional system may be differentiable. However, since the discretization schemes used in CFD codes are very complex, 'derivatives' and 'adjoint equations' should be treated with care and usually must be understood formally.

It is also important to keep in mind that the formulations (2.1)–(2.3) and (2.5), (2.6) are only equivalent if (2.2) has a unique solution $q(w)$ for all $w \in I\!R^{n_w}$ under consideration. If $R(q, w) = 0$ represents the (discretized) Euler equations, this assumption seems to be rather strong in view of the non-uniqueness result presented in [21] for discretized Euler equations. The existence and uniqueness of the solution q of $R(q, w) = 0$ for given w is often also related to the existence and uniqueness of the solution s_q of the linearized state equations $R_q(q, w)s_q + R_w(q, w)s_w + R(q, w) = 0$ for given $(q, w), s_w$.

As we have noted before, for a one-dimensional duct design problem, which is related to the airfoil design problem, the above issues have been rigorously discussed in [6]. For general airfoil design problems these issues are subject of current research. In our approach to the airfoil design problem we parameterize the airfoil using linear combinations of existing airfoils. This can be viewed as a reduced basis approach (see *e.g.*, [32, Sec.7.3]) leading to a low dimensional ($n_w = 4$) design space. Our grid generation scheme leads to grids which depend smoothly on the design parameters. Our application programs are based on the package ErICA for the simulation of flows over airfoils governed by the Euler equations. Among the discretization schemes available in that package, we use the schemes with better smoothness properties. We use the discretize-then-optimize approach. Since we have a low dimensional design space and a rather simple grid generation scheme, we believe this is sensible. However, given our experiences in [6], we believe this approach has to be rethought if more complex discretization schemes are used. More details on the discretization schemes are provided in Sections 3 and 6.

3 Analysis Problem, Discretization, and Flow Code

In this section we discuss the analysis problem underlying our design problem and its discretization. The analysis problem is the flow q around the airfoil governed by the steady state Euler equations for a perfect gas. We also outline the flow code ErICA used for the solution of the analysis problem. Although our optimization formulation is based on the all–at–once approach and our optimizer never needs to solve the Euler flow equations, we will extract several subtasks from the flow code ErICA. The presentation of the ErICA code will help to describe these tasks.

The unsteady Euler equations for a perfect gas, written in integral conservation law form is given by

$$\frac{\partial}{\partial t}\int_\Omega Q\, dS + \int_{\delta\Omega} \hat{F}\cdot\hat{n}\, ds = 0 \tag{3.1}$$

where, in Cartesian coordinates,

$$\hat{F} = \mathcal{F}\hat{j} + \mathcal{G}\hat{k}$$

and

$$Q = \begin{Bmatrix} \rho \\ \rho u \\ \rho v \\ \rho e_o \end{Bmatrix},\ \mathcal{F} = \begin{Bmatrix} \rho u \\ \rho u^2 + p \\ \rho uv \\ (\rho h_o)u \end{Bmatrix},\ \mathcal{G} = \begin{Bmatrix} \rho v \\ \rho uv \\ \rho v^2 + p \\ (\rho h_o)v \end{Bmatrix}$$

with velocity components u, v, density ρ, total energy per unit mass $e_o = e + (u^2+v^2)/2$, with e being the internal energy per unit mass and pressure p, which for a perfect gas may be expressed by the relation, $p = (\gamma-1)\rho e$. The total enthalpy per unit mass is given by $h_o = a^2/(\gamma-1) + u^2/2 + v^2/2 = e_o + p/\rho$, where $a = \sqrt{\gamma p/\rho}$ is the sonic velocity. Here, Q represents the conserved variables with $q = [\rho\ u\ v\ p]^T$ denoting the primitive variables, and, $\mathcal{F}$ and $\mathcal{G}$ represent the inviscid fluxes. The problem domain is denoted by Ω and $\delta\Omega$ represents the boundary of the domain. For a detailed treatment of the Euler equations refer [10].

3.1 Discretization of the Euler Equations

For a given airfoil configuration, the shape of which is represented as a function of the design variables w, the analysis problem corresponds to the solution of the Euler equations of flow. The flow is simulated numerically using the solver ErICA (EuleR Inviscid Code for Aerodynamics) which was developed by Narducci [25].

Computational simulations were performed on a C-type grid, which is wrapped around the airfoil. We only sketch the grid generation to fix some notation. For the omitted details we refer to [29, Sec. 4.2.2]. The grid is generated algebraically, by the following procedure: We first distribute points on bottom boundary, corresponding to the airfoil surface and the trailing edge wake, and on the top boundary of the computational grid, corresponding to the far-field boundary. Once the boundaries are defined, we connect corresponding pairs of points on the top and bottom boundaries using straight lines. Grid cells and nodes are numbered by (j,k), where j refers to the horizontal position and k refers to the vertical position in the grid. Indices with $k = 1$ refer to nodes or cells

on or at the airfoil, respectively. A typical 201×53 grid is shown in Figures 3.1 and 3.2 with 121 points on the airfoil surface. In practical CFD codes more sophisticated grid generation schemes are used. Eventually, such grid generations have to be incorporated. However, in a first attempt to apply the all–at–once methodology to airfoil design, we preferred this simple grid because of the relative ease of generating the grid and because of its guaranteed smooth dependence on the design parameters (airfoil).

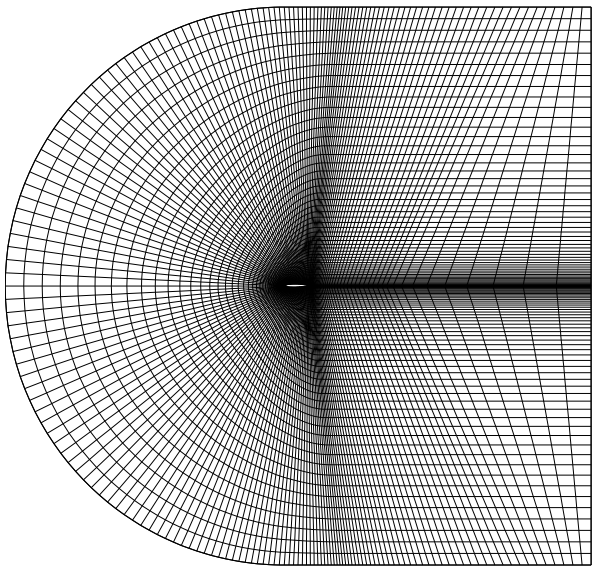


Figure 3.1: The 201×53 grid.

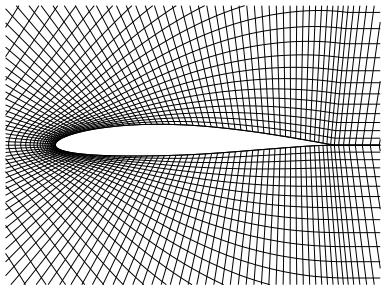


Figure 3.2: Close-up View of the 201×53 grid.

Given the grid, the ErICA code [25] is used for the solution of the steady state Euler equations.

In our version of ErICA a finite volume discretization using an upwind scheme with Van Leer Flux Vector Splitting is applied to compute the residuals. A MUSCL (Monotone Upstream-centered Scheme for Conservation Laws) differencing approach is used to interpolate the values of the state variables q from the cell centers to the cell faces. In order to fully capture the shock, we use third order interpolation of the fluxes. We use Van Albada's limiter to suppress oscillations in the flow solution. While other discretization schemes are available in ErICA, we selected these because of their better smoothness properties. See the discussion at the end of Section 2. A pseudo–time marching scheme is used to compute solution to the steady state Euler equations. We use the boundary conditions in [16, Ch. 19] on the airfoil surface and the far-field boundary conditions proposed in [30]. We describe the main features of the flow solver which are needed in the subsequent discussion. For more details we refer to [29].

As mentioned above, we use a cell-centered, finite volume formulation to rewrite the governing equations (3.1). For the (j,k)th grid-cell the (semi-discretized) residual in terms of the primitive variables is given by

$$S_{jk} M \frac{\partial q}{\partial t} + R_{jk}(q,w) = 0, \tag{3.2}$$

where S_{jk} is the area of the (j,k)th subdomain, $M = \frac{\partial}{\partial q} Q$ is the Jacobian of the mapping between the conserved and the primitive variables, and $R_{jk} = \sum_{sides} (\hat{F} \cdot \hat{n}) \Delta s$ is the residual, where $\hat{F}$ is the inviscid flux and Δs is the length of the side. Summation is done over all sides of cell (j,k). Requiring (3.2) for all cells yields

$$SM \frac{\partial q}{\partial t} + R(q,w) = 0. \tag{3.3}$$

The residual is computed as

$$R_{jk}(q,w) = \left[(\hat{F} \cdot \hat{n}) \Delta s\right]_{j-1/2} + \left[(\hat{F} \cdot \hat{n}) \Delta s\right]_{j+1/2} + \left[(\hat{F} \cdot \hat{n}) \Delta s\right]_{k-1/2} + \left[(\hat{F} \cdot \hat{n}) \Delta s\right]_{k+1/2}, \tag{3.4}$$

where $[\hat{F} \cdot \hat{n}]_{j \pm 1/2}$ correspond to the inviscid flux, $\hat{F} \cdot \hat{n}$, across the vertical cell faces, and $[\hat{F} \cdot \hat{n}]_{k \pm 1/2}$ corresponds to the flux across the horizontal cell faces, respectively. For a given cell face, we have,

$$\hat{F} \cdot \hat{n} = \left\{ \begin{array}{c} \rho U \\ \rho u U + \hat{n}_x p \\ \rho U v + \hat{n}_y p \\ (\rho h_0) U \end{array} \right\},$$

where $U (= \hat{n}_x u + \hat{n}_y v)$ is the velocity normal to the cell face, and $\hat{n}_x$ and $\hat{n}_y$ are the Cartesian components of the normal to the cell face. As noted above, the residual is computed using an upwind scheme, with Van Leer Flux Vector Splitting, with third order interpolation via MUSCL differencing to interpolate the values of the state variables, q, from the cell centers to the cell faces, and with Van Albada's limiter to suppress oscillatory behavior in the flow solution.

We use the surface boundary conditions described in [16, Ch. 19] and [8] (flow tangency stipulation and requirements that the normal momentum be zero and entropy be conserved, curvature

corrections described in [8] are neglected), along with the far-field boundary conditions proposed in [30]. These boundary conditions involve the creation of ghost cells.

The steady state solution corresponding to the semi-discrete equations (3.3) is computed using a pseudo–time marching scheme applied to (3.3). We consider the implicit scheme

$$\left[\frac{S}{\Delta t}M + \left(\widetilde{\frac{\partial R}{\partial q}}\right)^n\right]\Delta q = -R^n, \tag{3.5}$$

where $\Delta q = q^{n+1} - q^n$; $q^n = q(n\Delta t)$. Here $\widetilde{\frac{\partial R}{\partial q}}$ denotes an approximation of the Jacobian $\frac{\partial}{\partial q}R$. See below. The scheme (3.5) may be regarded as a simplified implicit Euler scheme since,

$$SM\frac{\Delta q}{\Delta t} = -R^{n+1} \approx -R^n - \left(\frac{\partial R}{\partial q}\right)^n \Delta q \approx -R^n - \left(\widetilde{\frac{\partial R}{\partial q}}\right)^n \Delta q.$$

The equation (3.5) is 'solved' by applying one step on an Alternating Direction Implicit (ADI) scheme. The resulting time–marching scheme is not accurate in time, but this is not required since we are only interested in steady state solutions.

For a moment, suppose that $(\widetilde{\frac{\partial}{\partial q}R})^n = (\frac{\partial}{\partial q}R)^n$ and that $\hat{A} = \frac{\partial}{\partial q}\hat{F}$. Then the equation of (3.5) corresponding to cell (j,k) is given by

$$\begin{aligned}\Big\{\frac{S}{\Delta t}M + \Big(&\Big[(\hat{A}\cdot\hat{n})\Delta s\Big]_{j-1/2} + \Big[(\hat{A}\cdot\hat{n})\Delta s\Big]_{j+1/2} \\ &+ \Big[(\hat{A}\cdot\hat{n})\Delta s\Big]_{k-1/2} + \Big[(\hat{A}\cdot\hat{n})\Delta s\Big]_{k+1/2}\Big)^n\Big\}\Delta q_{jk} \;=\; -R(q^n_{jk}).\end{aligned} \tag{3.6}$$

See (3.2), (3.4). Instead of using $\hat{A} = \frac{\partial}{\partial q}\hat{F}$, we make two simplifications to derive $\hat{A}$. These increase the efficiency with which one step of the pseudo–time marching scheme can be performed. The first simplification is as follows. In the residual computation, flux terms like $[(\hat{F}\cdot\hat{n})\Delta s]_{j+1/2}$ are calculated using Van Leer Flux Vector Splitting and MUSCL differencing with cubic interpolation of the values of the state variables q from the cell centers to the cell faces. ErICA analytically computes the Jacobians of the flux terms obtained using linear instead of cubic interpolation. The second simplification in computing $\hat{A}$ is made by partly suppressing the influence of the ghost-cells. Emphasizing the influence of the boundary conditions, the residual can be written as $R(q,w) = \check{R}(q,q_g(q),w)$, where q_g are the values of the flow variables on the ghost cells. The boundary conditions are used to express these as functions of the flow variables q in the interior and of w: $q_g = q_g(q,w)$. Thus, the derivative of the residual is of the form

$$\frac{\partial R(q,w)}{\partial q} = \frac{\partial \check{R}(q,q_g,w)}{\partial q} + \frac{\partial \check{R}(q,q_g,w)}{\partial q_g}\frac{\partial q_g(q,w)}{\partial q}. \tag{3.7}$$

In ErICA the approximation

$$\frac{\partial R(q,w)}{\partial q} \approx \frac{\partial \check{R}(q,q_g,w)}{\partial q} \tag{3.8}$$

is used to obtain $\hat{A}$.

Let $\hat{A} \approx \frac{\partial}{\partial q}\hat{F}$ denote the approximate flux-Jacobians derived using the two simplifications outlined above. If we set $\widetilde{A} = \frac{\widetilde{\partial R}}{\partial q}$, then $\widetilde{A}$ is a block pentadiagonal matrix. We split $\widetilde{A} = \widetilde{A}_k + \widetilde{A}_j$, where $\widetilde{A}_j$ corresponds to the terms $\left[(\hat{A}\cdot\hat{n})\Delta s\right]_{k-1/2} + \left[(\hat{A}\cdot\hat{n})\Delta s\right]_{k+1/2}$ in (3.6) and $\widetilde{A}_k$ corresponds to the terms $\left[(\hat{A}\cdot\hat{n})\Delta s\right]_{j-1/2} + \left[(\hat{A}\cdot\hat{n})\Delta s\right]_{j+1/2}$ in (3.6). The subscript j in $\widetilde{A}_j$ indicates that the matrix includes information of $\widetilde{A}$ along constant j–lines (vertical grid–lines). Similarly, $\widetilde{A}_k$ includes information of $\widetilde{A}$ along constant k–lines (horizontal grid–lines). We also define $T = \frac{1}{\Delta t}SM$. Now, (3.5) can be written as

$$\left[T + \widetilde{A}_k^n + \widetilde{A}_j^n\right]\Delta q = -R^n. \tag{3.9}$$

We do not solve (3.9), but approximately factor

$$T + \widetilde{A}_k^n + \widetilde{A}_j^n \approx \left[T + \widetilde{A}_k\right]^n T^{-1}\left[T + \widetilde{A}_j\right]^n$$

and solve

$$\left[T + \widetilde{A}_k\right]^n T^{-1}\left[T + \widetilde{A}_j\right]^n \Delta q = -R^n \tag{3.10}$$

The step Δq is computed by solving

$$\begin{aligned}\left[T + \widetilde{A}_k\right]^n \Delta q_{1/2} &= -R^n,\\ \left[T + \widetilde{A}_j\right]^n \Delta q &= T\,\Delta q_{1/2}.\end{aligned} \tag{3.11}$$

The new flow iterate is

$$q^{n+1} = q^n + \Delta q. \tag{3.12}$$

The two simplifications in the flux-Jacobians $\frac{\partial}{\partial q}\hat{F}$ leading to $\hat{A} \approx \frac{\partial}{\partial q}\hat{F}$ guarantee that (after symmetric permutation) the matrices on the left hand sides of (3.11) are block tridiagonal. Thus, each subproblem in (3.11) requires a block tridiagonal matrix inversion, which involves a block LU factorization and a block matrix solve; the latter consists of forward and backward substitutions. For more details about the scheme, we refer to Hirsch [17, 16] or Shenoy [29]. An outline of the ErICA algorithm for the solution of the governing Euler flow equations, $R(q, w) = 0$ for given w, is given in Algorithm 3.1.

3.2 Airfoil Shape Parameterization

We use an airfoil shape parameterization, formulated in [31]. See also [32, Sec. 10]. The airfoil geometry is represented as the weighted combination of six shape functions

$$y(x/c) = \sum_{i=1}^{6} \varpi_i y_i(x/c). \tag{3.13}$$

Four of the shape functions are pre-existing airfoils, namely, NACA 2412, NACA 64_1-412, NACA 65_2-415 and NACA 64_2-$A215$. The shape functions y_1–y_4 may be referenced from [1]. The two

Algorithm 3.1 (ErICA)

1 Given ϖ_i.

1.1 Generate C–grid, including ghost cells.

1.2 Compute direction cosines and lengths for each cell face and the areas of each cell.

2 Given q^n. Compute residual:

2.1 Impose Boundary Conditions.

2.2 Compute $R_{jk} = \sum_{\text{sides}} (\hat{F} \cdot \hat{n}) \Delta s$.

2.3 Compute $\|R\|$.

2.4 If $\|R\| < \text{tol}$, then output the result and stop; otherwise goto 3.

3 Euler Implicit Time Integration.

3.1 $k =$ constant lines:

Solve $\left[T + \widetilde{A}_k\right]^n \Delta q_{1/2} = -R^n$.

3.2 $j =$ constant lines:

Solve $\left[T + \widetilde{A}_j\right]^n \Delta q = T \Delta q_{1/2}$.

3.2 Update State: $q^{n+1} = q^n + \Delta q^n$. Set $n = n + 1$ and goto 2.

additional shape functions y_5, y_6 are used to impose certain geometric closure conditions at the trailing edge of the airfoil. These shapes are given by

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

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

Since these functions are used to close the airfoil at the trailing edge, the weights ϖ_5 and ϖ_6 are fixed in terms of ϖ_1–ϖ_4. We require that

$$y_{\text{u}s}(1) = y_{\text{l}s}(1) = 0,$$

which yield the following relations

$$\begin{aligned} \varpi_5 &= -[y_{1_{\text{u}s}}(1)\varpi_1 + y_{2_{\text{u}s}}(1)\varpi_2 + y_{3_{\text{u}s}}(1)\varpi_3 + y_{4_{\text{u}s}}(1)\varpi_4], \\ \varpi_6 &= [y_{1_{\text{l}s}}(1)\varpi_1 + y_{2_{\text{l}s}}(1)\varpi_2 + y_{3_{\text{l}s}}(1)\varpi_3 + y_{4_{\text{l}s}}(1)\varpi_4], \end{aligned}$$

where subscripts $\text{u}s$ and $\text{l}s$ refer to the upper and lower surfaces, respectively. An efficient approach to implement the above is to close each airfoil y_1–y_4 individually to obtain $\hat{y}_1$–$\hat{y}_4$, and use these as

our design bases. We have

$$y(x/c) = \sum_{i=1}^{4} \varpi_i \hat{y}_i(x/c). \tag{3.14}$$

4 The Design Problem

Given $\{\varpi_i\}$, the flow q around the airfoil is governed by the Euler equations for a perfect gas. The design problem is formulated as follows:

$$\max_{\varpi} C_L(q, \varpi) \tag{4.1}$$

such that

$$R(q, \varpi) = 0, \tag{4.2}$$
$$C_D(q, \varpi) \leq C_{D_{\max}}, \tag{4.3}$$
$$S_{\min} \leq S(\varpi) \leq S_{\max}, \tag{4.4}$$
$$\delta_{TE}(\varpi) \geq \delta_{\min}. \tag{4.5}$$

where $\varpi = \{\varpi_i\}$, and $q = [\rho \ u \ v \ p]^T$ denote the primitive variables of flow, with the usual notation. Equation (4.2) refers to the discretized steady state Euler equations of flow. The drag, C_D, in this case, is the wave drag. The lower limit on the area, S, is imposed so that the airfoil does not become too thin, a requirement for structural integrity. The upper limit is imposed to avoid thick, unrealistic airfoils. Equation (4.5) represents a bound on the trailing edge angle imposed to avoid situations in which the upper surface can go below the lower surface. See below. The free-stream conditions are based on Mach number $M = 0.75$ flow at angle of attack $\alpha = 0$.

The state equation (4.2) was discussed in Section 3. We briefly describe the computation of the aerodynamic forces used to compute $C_L(q, w)$ and $C_D(q, w)$, and the trailing edge condition (4.5). These are fairly standard, but are included for completeness.

The aerodynamic forces are computed by numerically integrating the pressure over the surface of the airfoil. The normalized forces normal and tangential to the airfoil chord line are respectively given by,

$$\begin{aligned} C_N &= -\frac{2}{\rho_\infty V_\infty} \sum_{j=j_o}^{j_f} p_{jk}(x_{j+1,1} - x_{j,1}), \\ C_T &= \frac{2}{\rho_\infty V_\infty} \sum_{j=j_o}^{j_f} p_{jk}(y_{j+1,1} - y_{j,1}), \end{aligned}$$

where (j, k), $k = 1$, $j = j_o, \ldots, j_f$, denote the grid points on the airfoil surface. We compute the lift and drag forces respectively as,

$$\begin{aligned} C_L &= C_N \cos\alpha - C_T \sin\alpha, \\ C_D &= C_N \sin\alpha + C_T \cos\alpha. \end{aligned} \tag{4.6}$$

The dependence of the lift and drag coefficients on the state q and the design ϖ can be determined from (4.6).

The area of the airfoil (for unit chordlength) is given by

$$S = \int_0^1 y_{\rm us} dx - \int_0^1 y_{\rm ls} dx.$$

Here $y_{\rm us}$, $y_{\rm ls}$ represent the upper and the lower surface, respectively. Representing the airfoil in terms of the basic (closed) airfoils (3.14), we have

$$\begin{aligned} S &= \int_0^1 \sum_{i=1}^4 \varpi_i \hat{y}_{i_{\rm us}} dx - \int_0^1 \sum_{i=1}^4 \varpi_i \hat{y}_{i_{\rm ls}} dx \\ &= \sum_{i=1}^4 \varpi_i \left(\int_0^1 \hat{y}_{i_{\rm us}} dx - \int_0^1 \hat{y}_{i_{\rm ls}} dx \right) = \sum_{i=1}^4 \varpi_i S_i. \end{aligned}$$

where S_i correspond to the areas of the individual airfoils. The areas of the individual airfoils can be computed at the beginning of the design cycle. For given ϖ the area is then simply computed as the weighted sum of the areas of the given (closed) airfoils.

Our parametric representation of the airfoil (3.14) allows for situations where the upper surface can go below the lower surface of the airfoil. Such situations were actually encountered in our preliminary attempts at optimization. Hence, we need to impose an additional constraint to prevent such physically incompatible configurations to arise. This is done by constraining the trailing edge angle of the airfoil. The trailing edge angle is given by $\tan^{-1}(y'_{\rm ls}(1)) - \tan^{-1}(y'_{\rm us}(1))$, which is approximated by

$$\delta_{TE} = y'_{\rm ls}(1) - y'_{\rm us}(1).$$

Using the (approximate) trailing edge angles δ_{TE_i} of the four basic airfoils, this can be written as

$$\delta_{TE} = \sum_{i=1}^4 \varpi_i \delta_{TE_i}. \tag{4.7}$$

It was found sufficient to impose the requirement (4.5) to ensure that the upper surface does not go below the lower surface. Note that while the above approximation of the trailing edge angle is fairly crude, it does yield a constraint that is easy to compute and achieves the desired effect.

5 Optimization Algorithm

The optimization algorithm used for our computation is a version of the trust-region interior-point SQP methods called TRICE developed in [9], [14], [15] for solving

$$\begin{aligned} \min \quad & J(q,w), && (5.1)\\ \text{s.t.} \quad & R(q,w) = 0, && (5.2)\\ & w_{\min} \le w \le w_{\max}. && (5.3) \end{aligned}$$

Clearly, (5.1)–(5.3) is a particular case of (2.1)–(2.3). In this section we give a brief description of the algorithm. We leave out many technical details and focus on how the algorithm interfaces with the flow solver. For more details on the algorithm and its convergence we refer to the papers [9], [14]. An important aspect of the TRICE implementation [15] is that application specific subtasks in the optimization are separated from the optimizer TRICE and can be provided by the user. In our case this allows us to provide approximate solutions to linearized state equations and adjoint equations computed by a modification of the flow code ErICA. See also Section 6. Since (5.3) corresponds to (2.3) with a G independent of q, the Lagrange multiplier λ is determined by (2.9). We use the equation (2.9) to define $\lambda = \lambda(q, w)$.

If we define a diagonal scaling matrix $D(q, w) \in I\!R^{n_w \times n_w}$ with diagonal elements

$$\Big(D(q,w)\Big)_{ii} = \begin{cases} (w_{\max} - w)_i^{\frac{1}{2}} & \text{if} \quad \Big(T(q,w)^T \nabla J(q,w)\Big)_i < 0, \\ (w - w_{\min})_i^{\frac{1}{2}} & \text{if} \quad \Big(T(q,w)^T \nabla J(q,w)\Big)_i \geq 0, \end{cases} \tag{5.4}$$

then the first order optimality conditions (2.8) can be equivalently written as

$$\begin{aligned} &R(q,w) = 0, \\ &D(q,w)^2 T(q,w)^T \nabla J(q,w) = 0, \end{aligned} \tag{5.5}$$

and $w_{\min} \leq w \leq w_{\max}$.

The algorithms in [9], [14], [15] generate a sequence of iterates (q_k, w_k), where w_k is strictly feasible with respect to the bounds, *i.e.*, $w_{\min} < w_k < w_{\max}$ (hence the term interior–point method). The algorithms can be motivated by applying Newton's method to the system of nonlinear equations (5.5) where the w component is kept strictly feasible with respect to the bounds, *i.e.*, $w_{\min} < w < w_{\max}$. The step $s = (s_q, s_w)$ is decomposed in to a quasi-normal step $s^{\mathtt{n}}$ and a tangential step $s^{\mathtt{t}}$. The role of the quasi-normal step $s^{\mathtt{n}}$ is to move towards feasibility. It is of the form $s^{\mathtt{n}} = (s_q^{\mathtt{n}}, 0)$. The q-component $s_q^{\mathtt{n}}$ is related to the Newton step applied to solve $R(q, w_k) = 0$, for given w_k. The role of the tangential step is to move towards optimality. It is of the form $s^{\mathtt{t}} = T(q_k, w_k) s_w = (-R_q(q_k, w_k)^{-1} R_w(q_k, w_k) s_w, s_w)$, where $T(q_k, w_k)$ is the representation of the null-space of the linearized state equation defined in (2.13). The w–component s_w of $s^{\mathtt{t}}$ is related to a quasi–Newton step for the reduced problem (2.5), (2.6).

The matrix $D(q, w)$ is in general not differentiable, but this nondifferentiability is benign and does not interfere with the fast convergence of Newton's method. A linearization of (5.5) around q_k, w_k gives

$$(R_q)_k s_q + (R_w)_k s_w = -R_k, \tag{5.6}$$

$$\Big(D_k^2 T_k^T \nabla^2_{(q,w)} \ell_k + [0 \mid E_k]\Big) \begin{pmatrix} s_q \\ s_w \end{pmatrix} = -D_k^2 T_k^T \nabla J_k. \tag{5.7}$$

Here we have used the subscript k to denote evaluation of functions at q_k, w_k. In (5.7), 0 denotes the $n_w \times n_q$ matrix with zero entries, $\ell(q, w, \lambda) = J(q, w) + \lambda^T c(q, w)$, $\nabla^2_{(q,w)} \ell(q, w, \lambda) = \frac{d}{d(q,w)}[J(q, w) + \lambda^T c(q, w)]$, and $E(q, w)$ is the $I\!R^{n_w \times n_w}$ diagonal matrix

$$\Big(E(q,w)\Big)_{ii} = \Big|\Big(T(q,w)^T \nabla J(q,w)\Big)_i\Big|$$

replacing the in general not existing term $[\frac{d}{d(q,w)}D^2(q,w)]T(q,w)^T\nabla J(q,w)$.

Since the solution of the linearized state equation (5.6) can be written as

$$s = s^{\mathrm{n}} + T_k s_w, \tag{5.8}$$

where $s^{\mathrm{n}} = (-(R_q)_k^{-1}R_k, 0)^T$ and T_k is given by (2.13).

By using (5.8) we can rewrite the linear system (5.6)–(5.7) as

$$s = s^{\mathrm{n}} + T_k s_w, \tag{5.9}$$

$$\left(D_k T_k^T \nabla^2_{xx}\ell_k T_k D_k + E_k\right) D_k^{-1} s_w = -D_k T_k^T \left(\nabla^2_{(q,w)}\ell_k s^{\mathrm{n}} + \nabla J_k\right), \tag{5.10}$$

The Newton–like step now is the solution of (5.9), (5.10) with D_k replaced by $\bar{D}_k$, where $\bar{D}_k$ is defined by (5.4) with $T_k^T\nabla J_k$ replaced by $T_k^T[\nabla^2_{(q,w)}\ell_k s^{\mathrm{n}} + \nabla J_k]$. This change of the diagonal scaling matrix is based on the form of the right hand side of (5.10).

One can that if (q_k, w_k) is close to a nondegenerate minimizer (q_*, w_*) which satisfies the second order sufficient optimality conditions, the matrix on the left hand side of (5.10) is positive definite. Therefore, (5.10) can also be interpreted as the optimality condition of a quadratic program in s_w. To globalize the convergence and to enhance robustness of the algorithm, a trust–region globalization is added. Let Δ_k be the trust radius at iteration k. The q–component of s^{n} is computed by approximately solving

$$\begin{array}{ll} \text{minimize} & \frac{1}{2}\|(R_q)_k(s^{\mathrm{n}})_q + R_k\|^2 \\ \text{subject to} & \|(s^{\mathrm{n}})_q\| \le \Delta_k. \end{array} \tag{5.11}$$

Given s^{n}, the step in w is computed by approximately solving

$$\begin{array}{ll} \text{minimize} & \left(T_k^T\left(H_k s_k^{\mathrm{n}} + \nabla J_k\right)\right)^T s_w + \frac{1}{2}s_w^T\left(T_k^T H_k T_k + E_k\bar{D}_k^{-2}\right)s_w \\ \text{subject to} & \|\bar{D}_k^{-1}s_w\| \le \delta_k. \end{array} \tag{5.12}$$

Of course, we also have to require that the new iterate is in the interior of the box constraints. To ensure that $w_k + s_w$ is strictly feasible with respect to the box constraints we choose $\sigma_k \in [\sigma, 1)$, $\sigma \in (0,1)$, and compute s_w with $\sigma_k(w_{\min} - w_k) \le s_w \le \sigma_k(w_{\min} - w_k)$. The quadratic minimization problems (5.11) and (5.12) only needs to be solved approximately. For example, an approximate solution of (5.11) is given by

$$(s^{\mathrm{n}})_q = -t[(R_q)_k]^{-1}R_k, \tag{5.13}$$

where $t = 1$ if $\|[(R_q)_k]^{-1}R_k\| \le \Delta_k$ and $t = \Delta_k/\|[(R_q)_k]^{-1}R_k\| \le \Delta_k$ otherwise.

An approximate solution of (5.12) can be computed using a modified conjugate gradient method. If the reduced Hessian $T_k^T H_k T_k$ is approximated by a quasi-Newton update, then the solution of (5.12) is relatively inexpensive. The cost of computing the 'reduced' gradient $T_k^T(H_k s_k^{\mathrm{n}} + \nabla J_k)$ dominates the cost of solving (5.12).

The main steps of the trust–region interior–point SQP scheme are outlined in algorithm 5.1. More general versions of these algorithms are possible. See [9], [14], [15].

Algorithm 5.1 (TRICE)

1 Given H_0 and Δ_0.

2 For $k = 0, 1, 2, \ldots$ do

2.1 Compute $s_k^{\mathtt{n}}$ by approximately solving (5.11).

2.2 Compute $T_k^T(H_k s_k^{\mathtt{n}} + \nabla J_k)$.

2.3 Compute s_w with $\sigma_k(w_{\min} - w_k) \leq s_w \leq \sigma_k(w_{\max} - w_k)$ by approximately solving (5.12).

2.4 Compute $s = s^{\mathtt{n}} + s^{\mathtt{t}} = s^{\mathtt{n}} + T_k s_w$.

2.5 Compute λ by solving (2.9) with $q = q_k + s_q$, $w = w_k + s_w$.

2.6 Update the trust region radius Δ_k and decide if $q_k + s_q$, $w_k + s_w$ can be accepted as the new iterate.

2.7 If s is rejected set $q_{k+1} = q_k$, $w_{k+1} = w_k$, and $\lambda_{k+1} = \lambda_k$.
Otherwise, s is accepted; set $q_{k+1} = q_k + s_q$, $w_{k+1} = w_k + s_w$ and $\lambda_{k+1} = \lambda$.

2.8 If exact second order information is not used, update the (reduced) Hessian approximation.

We briefly sketch the information that the SQP algorithm 5.1 requires from the application programs. A detailed presentation is given in [15]. If (5.13) is used, then step 2.1 requires the solution of a linearized state equation. The computation in step 2.2 involves the solution of an adjoint equation, see the definition (2.13) of $T(q, w)$. If a quasi–Newton approximation is used to replace the the reduced Hessian $T_k^T H_k T_k$, then step 2.3 can be implemented very efficiently using a modified conjugate gradient method. The application of T_k in step 2.4 requires the solution of another linearized state equation. See (2.13). Computations involving the solution of one linearized state equation and one adjoint equation may be needed in step 2.8. This depends on the update used. This and other algorithms are implemented in [15]. Global and local convergence results are proven in [9]. The influence of inexact derivatives is analyzed in [14]. The latter aspect is important in our application since we use a pseudo–time marching (iterative) scheme to compute approximate solutions to linearized state equations and to the adjoint equations. Moreover, additional approximations, outlined in Section 3, are applied in these computations as well.

A final remark on the handling of inequality constraints is in order. In our case the design space is small $n_w = 4$ and other approaches such as projection methods or active set methods can likely be used with similar performance to handle the inequality constraints (5.3). However, wing designs in industrial settings may involve up to 500 design parameters [4]. In this case an interior point approach promises to be superior.

6 Numerical Implementation

6.1 Handling the Inequality Constraints and Reformulation of the Optimization Problem

The current version of TRICE only solves problems of the form (5.1)–(5.3). Hence we need to recast the design problem (4.1)–(4.4) into the form (5.1)–(5.3). This is done by handling (4.3) as a "soft" constraint using a penalty term and by transforming the design parameters.

Instead of including the drag constraint (4.3) we add a penalty term $P(\varrho(C_D/C_{D_{max}} - 1))$ to the objective, where

$$P(z) = \begin{cases} 0 & z \leq 0, \\ z^2 & z > 0. \end{cases} \tag{6.1}$$

Here ϱ is a (scalar) penalty constant which can be used to increase emphasis on the drag violation. The addition of $P(z)$ to the objective function J has the desired effect of penalizing the objective when the drag constraint is violated. Note that this is a "soft" constraint, in the sense that the optimizer will allow the drag constraint (4.3) to be violated as long as the penalty term added is not too large. This can be addressed to some extent by controlling the penalty constant ϱ.

The remaining constraints can be addressed using a mapping between the control variable, w, and the design weights, ϖ. Rather than use the design weights as our control variables, we use the area of the airfoil and its trailing edge angle as control variables, as shown below. This enables us to address the issue of ensuring that the lower bound on the area and the trailing edge angle remain strictly enforced, by making use of the fact that we can place bounds on the control variables. We use, as our control variables,

$$w = W \begin{Bmatrix} 2 \cdot (\varpi_1 - \varpi_3) \\ 2 \cdot (\varpi_2 - \varpi_4) \\ 2 \cdot \bar{S} \\ 0.5 \cdot \bar{\delta}_{TE} \end{Bmatrix} \tag{6.2}$$

where

$$\bar{S} = \frac{S}{S_{min}}, \qquad \bar{\delta}_{TE} = \frac{\delta_{TE}}{\delta_{min}},$$

and the scalar factor W has been added in order to be able to experiment with the scaling. Note that this results in a control space that is simply a result of a linear combination of the design weights, as should be evident from equations (4.7) and (4.7). Now, enforcing bounds

$$2W \leq w_3 \leq 2W \cdot \frac{S_{max}}{S_{min}}$$

strictly enforces the bounds on the area of the airfoil (4.4). Similarly, the bound,

$$0.5W \leq w_4$$

imposes the constraint (4.5). The mapping (6.2) can be used to translate between w and ϖ. Note that the above mapping was chosen so as to yield a low condition number for the transformation

matrix providing the mapping between the weights and the controls. This was done with the philosophy that a change in the controls should produce roughly the same amount of change in the weights. This choice did yield improved performance in the optimization algorithm.

The original design problem (4.1)–(4.4) is now recast as

$$\begin{aligned} \min_{q,w} \quad & J(q,w) = -C_L(q,w) + P(\varrho(C_D(q,w)/C_{D_{max}} - 1)) \\ \text{s.t.} \quad & R(q,w) = 0, \\ & w_{\min} \le w \le w_{\max}, \end{aligned}$$

which is of the form (5.1)–(5.3).

6.2 Solution of the Linearized State and Adjoint Equations

To solve the design problem using Algorithm 5.1 described above, we need to be able to do the following:

- Provide an update s_q in the state variable, given an update in the control variable s_w by solving the linearized state constraint.
- Solve the adjoint equation at a given point.

These tasks can be performed using the same problem-solving structure applied in the flow solver ErICA. These tasks can be easily extracted from the flow code and only relatively few changes are needed.

The modification of the ErICA code to compute approximate solutions to the linearized state equation is shown in Algorithm 6.1. The scheme outlined in Algorithm 6.1 solves an approximation

$$\left(\widetilde{A}_j + \widetilde{A}_k + \widetilde{A}_g\right) s_q + \frac{\partial R}{\partial w}(q,w) s_w + R(q,w) \equiv \bar{R}(s_q, s_w, q, w) = 0 \tag{6.3}$$

of the linearized Euler equations using a pseudo-time marching scheme analogous to the one applied in ErICA. Here q, w and s_w are given and an approximate solution s_q has to be computed. Note that $\frac{\partial}{\partial q} R(q,w)$ is replaced by $\widetilde{A}_j + \widetilde{A}_k + \widetilde{A}_g$. While in the residual computation, flux terms are calculated using Van Leer Flux Vector Splitting and MUSCL differencing with cubic interpolation of the values of the state variables q from the cell centers to the cell faces, only linear interpolation is used to calculate approximate Jacobians, see Section 3. This leads to the matrices $\widetilde{A}_j$, $\widetilde{A}_k$. However, boundary conditions are included in the residual computations, *cf.* (3.7). This is reflected above by the matrix $\widetilde{A}_g$. The equation (6.3) is solved by driving an *unsteady* form of the linearized Euler equations

$$SM \frac{\partial s_q}{\partial t} = -(\widetilde{A}_j + \widetilde{A}_k + \widetilde{A}_g) s_q - \frac{\partial R}{\partial w}(q,w) s_w - R(q,w) \equiv -\bar{R}(s_q, s_w, q, w) = 0 \tag{6.4}$$

towards steady state. The factor SM is added to the transient term in order to make the above equation consistent with the discretized Euler equations (*cf.* (3.3)). The pseudo-time marching

scheme used is identical to the one used in marching the nonlinear Euler equations, described above in equations (3.10)–(3.12), with the nonlinear residual, R replaced by the linearized residual, $\bar{R}$. We can view this algorithm as simply an iterative method for solving the linearized state equation. Note that a relaxation factor β is used to update the solution in the iterative process. Our numerical experiments showed that using $\beta = 1.25$ yielded improved convergence rates. Also, note that there is an external loop in the iterative process monitoring the residual. This is in order to ensure that the residual does not diverge. If the norm of the residual is greater than some predetermined value, $\bar{R}_{\max}$, then the iterative process is restarted with a reduced time step Δt. This is necessitated by the fact that the Jacobians can be ill-conditioned if the solution is far from feasible, resulting in a divergent iteration. Reducing the time step had the effect of alleviating the ill-conditioning. Using $\bar{R}_{\max} = 12\|\bar{R}^0\|$ seemed adequate for our purposes. A factor of 12 is used because in some instances, the iterative process initially increased the residual but managed to recover. Clearly, these rules are somewhat ad-hoc and more sophisticated techniques could have been applied to increase efficiency. Since we are concerned with more fundamental issues arising in the all-at–once approach, optimizing performance is beyond the scope of this study. Note that Algorithm 6.1 only involves one Jacobian evaluation and one nonlinear residual evaluation. The Jacobian undergoes one block LU factorization and the iterative loop only involves block matrix solves, and evaluation of the linearized residual, which simply requires relatively cheap block matrix multiplications and additions.

Similarly, for the solution of the approximate adjoint equation

$$(\widetilde{A}_j + \widetilde{A}_k + \widetilde{A}_g)\lambda + +\nabla_q J(q,w) \equiv \Lambda(q,w) = 0 \tag{6.5}$$

consider a "pseudo" time dependent adjoint equation,

$$SM^T \frac{\partial \lambda}{\partial t} = -\left(\left(\widetilde{A}_j + \widetilde{A}_k + \widetilde{A}_g\right)^T \lambda + \nabla_q J(z_k)\right) \equiv -\Lambda,$$

where J is the objective. As in the solution of the linearized state equation, we replace $\frac{\partial}{\partial q}R$ by $\widetilde{A}_j + \widetilde{A}_k + \widetilde{A}_g$. We use the approximate factorization algorithm used in ErICA to iterate this equation in time, until the residual of the adjoint equation is sufficiently small, ideally $\Lambda = 0$. The procedure is as follows. We have,

$$\left[T + \widetilde{A}_j + \widetilde{A}_k\right]^T \Delta\lambda = -\Lambda^n$$

where $\widetilde{A}_j$ and $\widetilde{A}_k$ are Jacobian terms. The matrix of the left is factored approximately according to spatial directions

$$\left[T + \widetilde{A}_j\right]^T T^{-T} \left[T + \widetilde{A}_k\right]^T \Delta\lambda = -\Lambda^n$$

This system is solved using the sequence

$$\begin{aligned} \left[T + \widetilde{A}_j\right]^T \Delta\lambda_{1/2} &= -\Lambda^n \\ \left[T + \widetilde{A}_k\right]^T \Delta\lambda &= T^T \Delta\lambda_{1/2} \end{aligned} \tag{6.6}$$

Algorithm 6.1 (Linearized State Equation Solver)

1 Given q, w, s_w, tol.

1.1 Generate grid.

1.2 Compute: $\widetilde{A}_j(q,w)$, $\widetilde{A}_k(q,w)$, $\widetilde{A}_g(q,w)$, $R_w(q,w)$.

1.3 Compute: $\bar{R}^0 = R(q,w) + R_w(q,w)s_w$.

1.4 Set: $n = 0$, ${s_q}^n = 0$.

2 LU Decomposition.

2.1 Compute: $L_k U_k = \left[T + \widetilde{A}_k\right]$.

2.2 Compute: $L_j U_j = \left[T + \widetilde{A}_j\right]$.

3 Euler Implicit Time Integration.

3.1 $k =$ constant lines:

Solve $L_k U_k \Delta s_{q_{1/2}} = -\bar{R}^n$.

3.2 $j =$ constant lines:

Solve $L_j U_j \Delta s_q = T \cdot \Delta s_{q_{1/2}}$.

3.2 Update s_q: $s_q^{n+1} = s_q^n + \Delta s_q$. Set $n = n + 1$ and goto 4.

4 Compute Linearized Residual

4.1 Compute: $\bar{R}^n = \left[\widetilde{A}_j + \widetilde{A}_k + \widetilde{A}_g\right] {s_q}^n + \bar{R}^0$

4.2 Compute: $\|\bar{R}^n\|$.

If $\|\bar{R}^n\| <$ tol, set $s_q = s_q^n$. Return.

Else, if $\|\bar{R}^n\| \leq \bar{R}_{\max}$, goto 3.

Else, restart. Set: $\Delta t = 0.5\Delta t$, $n = 0$, ${s_q}^n = 0$, and goto 2.

$$\lambda^{n+1} = \lambda^n + \beta\Delta\lambda$$

The above iteration is performed until the residual of the adjoint equation, Λ, is reduced to zero. The adjoint computation is outlined in Algorithm 6.2. Note, that while computing the residual of the adjoint equation, and the linearized state equation, we include the terms $\widetilde{A}_g$ corresponding to the boundary conditions.

Since we are interested in the solution of the steady state adjoint equation (6.5), we could have just as well reversed the sequence above, *i.e.*, solve

$$\begin{aligned} \left[T + \widetilde{A}_k\right]^T \Delta\lambda_{1/2} &= -\Lambda^n \\ \left[T + \widetilde{A}_j\right]^T \Delta\lambda &= T^T \Delta\lambda_{1/2} \end{aligned} \tag{6.7}$$

and set $\lambda^{n+1} = \lambda^n + \beta\Delta\lambda$. This would give us an algorithm which is exactly the same as that used in the linearized state algorithm, except the matrices would be transposed. However even though the pseudo-time marching is just an iterative scheme for solving (6.5), we preferred to use the transpose of the pseudo-time process (6.4) to calculate the adjoints. Numerical experiments showed that (6.6) had a slightly superior convergence behavior than (6.7).

Once again an outer loop monitors divergence of the residual. We use $\Lambda_{\max} = 12\|\Lambda^0\|$. Numerical experiments showed that the computation of the adjoint was more susceptible to producing divergent results, and hence care has to be taken in choosing the value of the relaxation factor β. We choose $\beta = \min(1.25, 1 - 0.12\log(10\|\Lambda^i\|))$, which has the desired effect of underrelaxing when the solution is crude, in order to reduce the possibility of divergence, and overrelaxing when the solution is refined in order to increase speed of convergence. Also, note that we do not start with $\lambda = 0$. Rather, we start from the previously computed estimate of the adjoint variable. Our experiments showed that this yielded significant savings in terms of the number of iterations. However, if the iteration proves to be divergent, then we reset λ to zero. As we have noted already for the linearized state solver, these rules are somewhat ad-hoc and more sophisticated techniques could have been applied to increase efficiency. This will be done in future studies. As with the procedure for the linearized state equation, this iteration only involves a single Jacobian evaluation.

7 Numerical Results and Discussion

This section reports on some of numerical experiments conducted using the TRICE interior-point trust-region SQP optimization algorithm (see Section 5) coupled with the modification of the ErICA flow code (see Sections 3 and 5) to solve the airfoil design problem stated in Sections 4 and 6. The presentation of results is followed by a discussion of observed difficulties, possible remedies, and further research issues.

All reported computations were performed on C–type grids with the following dimensions:

1. 51×14 grid with 31 points on the airfoil surface,
2. 101×27 grid with 61 points on the airfoil surface,
3. 151×40 grid with 91 points on the airfoil surface,
4. 201×53 grid with 121 points on the airfoil surface,
5. 301×79 grid with 181 points on the airfoil surface.

Algorithm 6.2 (Adjoint Equation Solver)

1 Given q, w, tol.

1.1 Generate grid.

1.2 Compute: $\widetilde{A}_j(q,w)$, $\widetilde{A}_k(q,w)$, $\widetilde{A}_g(q,w)$, $R_w(q,w)$.

1.3 Compute: $\Lambda^0 = \nabla_q J(q,w)$.

1.4 Set: $n = 0$, $\lambda^n = \lambda_{\rm prev}$.
($\lambda_{\rm prev}$ is the Lagrange multiplier estimate at the previous iteration)

2 LU Decomposition.

2.1 Compute: $L_k U_k = \left[T + \widetilde{A}_k\right]$.

2.2 Compute: $L_j U_j = \left[T + \widetilde{A}_j\right]$.

3 Compute Adjoint Residual

3.1 Compute: $\Lambda^n = \left[\widetilde{A}_j + \widetilde{A}_k + \widetilde{A}_g\right]^T \lambda^n + \Lambda^0$

3.2 Compute: $\|\Lambda^n\|$.
If $\|\Lambda^n\| <$ tol, set $\lambda = \lambda^n$. Return.

Else, if $\|\Lambda^n\| > \Lambda_{\max}$, restart. Set: $\Delta t = 0.5\Delta t$, $n = 0$, $\lambda^n = 0$, goto 2.

4 Euler Implicit Time Integration.

4.1 $j =$ constant lines:
Solve $L_j U_j \Delta\lambda_{1/2} = -\Lambda^n$.

4.2 $k =$ constant lines:
Solve $L_k U_k \Delta\lambda = T \cdot \Delta\lambda_{1/2}$.

4.3 Update Adjoint: $\lambda^{n+1} = \lambda^n + \Delta\lambda$. Set $n = n + 1$ and goto 3.

Before running the optimization, we used ErICA for simulation and performed a grid convergence study. This was done for the (closed) NACA 2412 airfoil and the grids specified above. The results are shown in Figure 7.1. They show that the analysis code ErICA overestimates the drag C_D and underestimates the lift C_L on the coarser grids.

The grid convergence study motivated the following procedure for solving the design problem. We solve the design problem on a sequence of grids, using the optimal solution (design parameters w and interpolation of corresponding flows q) of the coarse grid as the initial value on the next finer grid. On the coarse grid we compute starting values as follows: The initial design parameters $w = (1, 0, 0, 0)$ correspond to the (closed) NACA 2412 airfoil and the initial flow q was the corresponding flow computed by ErICA on the coarse grid, *i.e.*, we start with a point satisfying the state

equation $R(q, w) = 0$. We also start with a loose bound $C_{D_{\max}}$ on the drag, which is tightened gradually as we refine the grid. The values are reported in Table 7.1. We use $\varrho = 10$ in the penalty term for the drag and $S_{\min} = 0.075$ and $S_{\max} = 0.15$.

We stop the optimization on the current grid if the norm of the reduced gradient $T(q, w)^T \nabla J(q, w)$ is reduced to a tolerance of 10^{-2} and if $\|R(q, w)\| \leq 10^{-5}$. It was observed that the optimizer struggled to diminish the reduced gradient much below this. For the 51×14 grid, a scale factor of $W = 1$ was used, which produced results that had reduced the norm of the reduced gradient to about 1.5×10^{-2}, at which point the iteration was terminated due to lack of progress. The algorithm was implemented so that if the optimization stalled, *i.e.*, the algorithm terminated because it failed to make further progress, we restart the optimization with a feasible solution for the given configuration. Several restarts were required for the 51×14 grid. In particular, the optimizer struggled if the constraint residual $\|R(q, w)\|$ became too large, *i.e.*, if the iterates move too far away from feasibility. The observed difficulties in the optimization are likely due to the inaccuracies of residual Jacobians used in the linearized state equations and the adjoint equations. We will discuss this in more detail below. For the other grids we used a scale factor of $W = 10$ (*cf.* (6.2)). The optimization converged with a single restart in these cases. The results are shown in Table 7.1. Note that we did not run the optimizer for the final grid, as we had a nearly converged solution. The area of the optimized airfoil is at the lower bound, as one might expect.

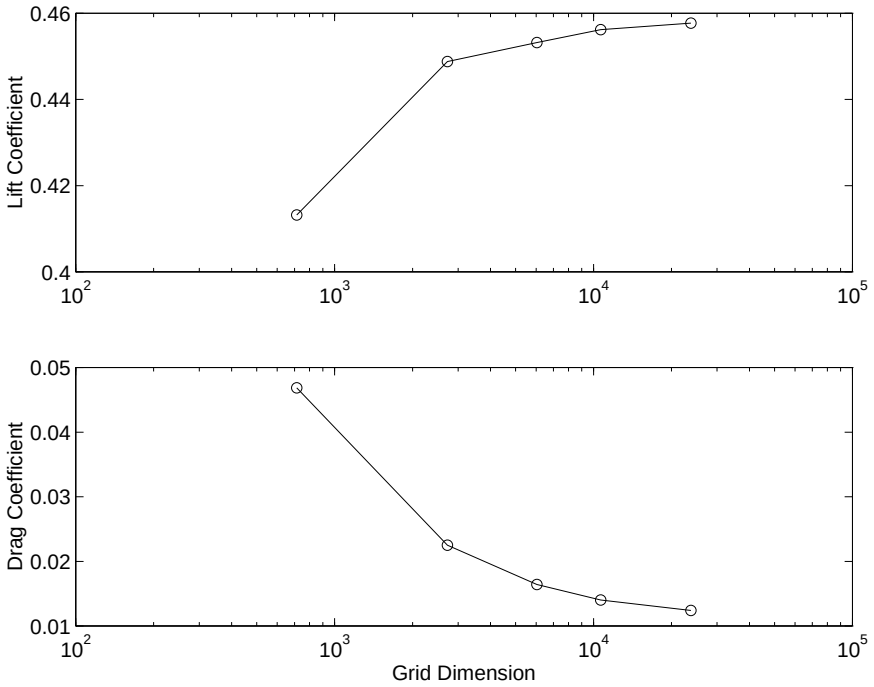


Figure 7.1: Grid Convergence Study for NACA 2412

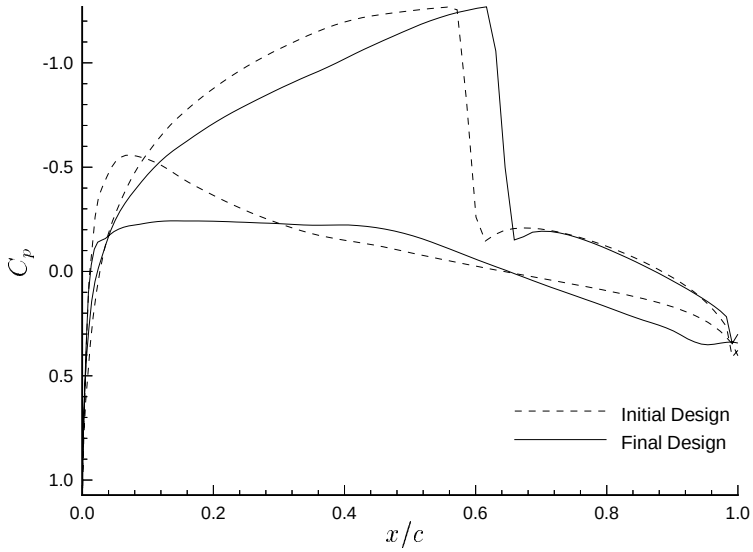
Figure 7.2 shows the final results obtained for the 301×79 grid. Note that the shock has moved downstream. The optimization improves the performance of the airfoil. Compared to the baseline NACA 2412 airfoil, which had a lift coefficient, $C_L = 0.4577$ with $C_D = 0.01241$, the

Table 7.1: Numerical Results for Airfoil Design Problem

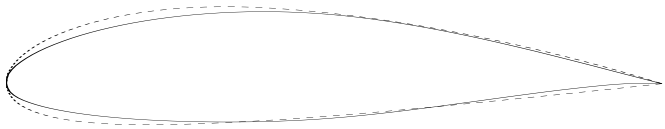
Grid Size	$C_{D_{max}}$	Data	ϖ_1	ϖ_2	ϖ_3	ϖ_4	C_L	C_D	S
51×14	0.04	Initial	1.0000	0.0000	0.0000	0.0000	0.4132	0.0469	0.0823
		Final	0.2538	0.1793	-0.1369	0.5793	0.5263	0.0403	0.0770
101×27	0.02	Initial	0.2538	0.1793	-0.1369	0.5793	0.5722	0.0251	0.0770
		Final	0.2812	0.1049	-0.0760	0.5314	0.5227	0.0201	0.0750
151×40	0.014	Initial	0.2812	0.1049	-0.0760	0.5314	0.5266	0.0159	0.0750
		Final	0.3059	0.0593	-0.0313	0.5006	0.5037	0.0142	0.0750
201×53	0.012	Initial	0.3059	0.0593	-0.0313	0.5006	0.5044	0.0126	0.0750
		Final	0.3153	0.0429	-0.0154	0.4893	0.4955	0.0120	0.0750
301×79	0.010		0.3153	0.0429	-0.0154	0.4893	0.4959	0.0103	0.0750

final design has $C_L = 0.4959$ and $C_D = 0.01028$, which means the lift coefficient has increased by approximately 8.5%, while the drag coefficient has been reduced by 17%.

Table 7.2 gives an account of the computational effort required at each step to produce a "converged" solution. However, one should keep in mind that we did not optimize the implementation for efficiency. Improvements in performance can be achieved and we outline a few possible enhancements below. Here the number of "successful" iterations is the number of iterations in which the Algorithm 5.1 accepts the step s and uses $q_{k+1} = q_k + s_q$, $w_{k+1} = w_k + s_w$ as the new iterate (see step 2.7), whereas the total iterations also includes the unsuccessful iterations, *i.e.*, those in which the step is rejected and the next iterate is set to be $q_{k+1} = q_k$, $w_{k+1} = w_k$. It should be noted that most of the computational effort (in terms of number of iterations) occurs at the coarsest level, where the computations are fairly cheap. Though there is room for improvement the TRICE algorithm is relatively efficient in finding the given solutions. Consider, for example, the computational effort required for the 51×14 grid. We require 3786 residual evaluations, 5004 Jacobian evaluations, 10008 block LU factorizations and 454948 block matrix solves. Compare this to the effort required to obtain a single analysis solution: We require approximately 1000 pseudo–time integration steps to produce a converged solution which requires 1000 residual evaluations, 1000 Jacobian evaluations, 2000 block LU factorizations and 2000 block matrix solves. Discounting the discrepancy in the number of solves, the computational effort required by TRICE is roughly equal to the effort required to perform 5–6 flow analyses, which is very cheap. It should be noted that though we require a large number of block matrix solves, this just involves forward and backward substitutions which is fairly cheap. As we have indicated earlier, this paper concerned with the feasibility of the all–at–once approach for airfoil design. Computational efficiency was not a



a. Surface Pressure Distribution



b. Initial airfoil (dashed) and optimal airfoil (solid)

Figure 7.2: Results obtained using TRICE for Airfoil Design Problem

prime concern for the interface of the ErICA flow subroutines with the TRICE optimizer. Several improvements can be made. For example, we recompute the Jacobian information every time that we require to solve the linearized state equation or the adjoint equation. Since the Jacobian will only change if the iterate (q, w) changes, a more efficient implementation would require only one Jacobian evaluation for each iteration of Algorithm 5.1. Moreover, our pseudo-time-stepping scheme for solving the linearized state and the adjoint equation can be improved which would lead to fewer pseudo-time-steps and fewer LU solves. Similar instances of implementing the code in a more efficient manner should yield significant savings from the elimination of redundant computations.

Earlier, we have pointed out that the optimizer struggled to achieve the desired tolerances. We now discuss possible reasons for this and also possible remedies. We distinguish among three groups. The first group is related to the information that is provided to the optimizer, the second group is related to the optimization formulation, and the third group is related to the airfoil design problem and its discretization.

The TRICE optimizer requires from the user the solution of linearized state equation and ad-

Table 7.2: Computational History for Airfoil Design Problem using TRICE

Grid Size	Total Iterations	Successful Iterations	Number of Restarts	# Residual Evaluations	# Jacobian Evaluations	# LU Factorizations	# Solves
51×14	388*	211	12	3786	5004	10008	454948
101×27	44	26	1	313	437	874	73588
151×40	27	11	1	506	518	1036	81953
201×53	19	5	1	386	425	850	137768

* Iteration was stopped with a norm of the reduced gradient of $1.5 \cdot 10^{-2}$ due to lack of progress.

joint equation. We extract this information from the ErICA code. However, we use simplifications in the computation of the constrain residuals. While the residual $R(q, w)$ is evaluated using Van Leer Flux Vector Splitting and MUSCL differencing with cubic interpolation of the values of the state variables q from the cell centers to the cell faces, the 'Jacobians' of the flux terms are obtained using linear instead of cubic interpolation. Thus we only use approximate Jacobians. These approximations become better as the grid is refined, but on a given grid a certain error level cannot be removed. This explains our earlier observation that the optimizer had more problems finding a solution relative to the given tolerances on the coarse grid than on the fine grid. The Jacobians used are only asymptotically correct and the discrepancies between true Jacobians and Jacobians used become smaller as the mesh is refined. On coarse grids, we try to oversolve the problem. We expect a significant improvement in performance if true Jacobians are used. However, in that case some complications may arise from the differentiation of the Van Albada flux limiter. This matter will be investigated in depth in future research. We point out that this behavior does not contradict the ability of the optimizer to handle inexact information. The optimizer can only perform successfully for arbitrary stopping tolerances if the degree of inexactness can be adjusted by the optimizer to the progress it makes towards computing the solution and thereby to the required tolerance. One needs to adjust the stopping tolerances to the accuracy in function values. In fact, we could have relaxed the tolerance on the coarse grid and thereby reduced the number of coarse grid iterations, while maintaining the performance on the finer grids. However, since an exact error bound for the quality of the Jacobians used is not available, we believe that one tends to try to oversolve the problem. Hence, the performance displayed in Table 7.2 is what one should expect in the experimentation phase of the algorithm. These experiments can be used to define grid-dependent tolerances which will lead to a better performance than that shown in Table 7.2.

We used the optimizer TRICE because of its capability to accept solutions of linearized state equations and adjoint equations computed using application specific solvers. A reformulation of the problem presented in Section 6 was necessary, since the current version of TRICE only solves problems of the form (5.1)–(5.3). Some inefficiencies and difficulties might be attributed to this. It is expected that these will be resolved with future version of the optimizer for solving the more

general problems (4.1)-(4.5). A high percentage of the number of block matrix solves required to solve the design problem (refer Table 7.2) can be attributed to the large number of iterations required to obtain a converged solution for the adjoint equation. This is especially true for the finer grids, which may be caused by ill-conditioning in the grid. We return to this issue below. The high number of solves can also be partly attributed to the penalty function approach we use to address the drag constraint (6.1). When the drag constraint is violated the gradient of the objective function with respect to the state variables becomes very large, which in turn means that the residual of the adjoint equation is very large and requires a large number of iterations to converge. The penalty term (6.1) also causes objective function to change abruptly when the drag constraint is violated. This effect is currently inadequately reflected in the model (5.12) used to compute the step and led to a large number of unsuccessful iterations.

The third group of reasons for difficulties in the solution process is somewhat related to the first one and concerns the airfoil design problem and its solution. Jameson [21] has shown the existence of cases where nonunique solutions of the discretized Euler equations can be obtained for certain airfoils. While the all–at–once approach never requires the solution of the Euler equation, our implementation uses the solution to the linearized equations and adjoint equations. Existence of these solutions and their dependence upon right hand side data need to be investigated. Another possible reason for the difficulties in convergence behavior is the fact that the simple algebraic grid that we use may be ill-conditioned. Since the primary purpose of the present study was to demonstrate the concept of using the all–at–once approach for solving the design problem, we have not investigated the effect of the grid on the solution process. As a result, no attempt has been made to ascertain the quality of the grid. It was, in fact, observed that the convergence behavior exhibited by the flow solver deteriorates as the grid is refined, which could be an indication of ill-conditioning. Other grid generation techniques and airfoil surface discretizations should be investigated in this context. More sophisticated techniques are applied in many of the papers on airfoil design cited earlier in this paper. However, an inclusion of such techniques in the all-at–once approach requires a careful analysis of grid sensitivities which are needed in the computation of $R_w(q, w)$ and $J_w(q, w)$. Finally, as we have pointed out in Section 2, the airfoil design problem is an infinite dimensional problem. The infinite dimensional problem, its discretization, and the optimization approach have to be analyzed jointly to derive robust and efficient solution methods. In a simplified model problem the benefits of such an analysis were demonstrated in [6] and were shown to lead to 10–15% reductions in optimization iterations. Provisions for the inclusion of infinite dimensional problem structure into the optimizer have been made, see [15]. Their application in the airfoil design problem are part of future research.

8 Conclusion

We have implemented the all–at–once approach to solve an optimum airfoil design problem. The airfoil design problem was formulated as a constrained optimization problem in which flow variables and design variables are viewed as independent variables and in which the coupling steady state 2-D Euler equation is included as a constraint. To implement this approach, we have com-

bined an existing optimization algorithm, TRICE, with an existing flow code, ErICA. Details of the implementation were given and difficulties arising in the implementation were discussed. Our numerical results indicate that the cost of solving the design problem is approximately six times the cost of solving a single analysis problem. This is consistent with the expectation that the decoupling of flow variables and design variables in the all–at–once approach makes the problem less nonlinear and can increase the efficiency with which the design problem is solved. Difficulties observed in the solution process were discussed and some future research issues were addressed.

Acknowledgements

We would like to thank Robert Narducci and Prof. Bernard Grossman for making available to us the flow solver ErICA, and for their assistance in extracting and understanding the information generated through the flow simulations.

References

[1] I. H. ABBOTT AND A. E. VON DOENHOFF, *Theory of Wing Sections*, Dover Publications, Inc., New York, 1959.

[2] W. K. ANDERSON AND V. VENKATAKRISHNAN, *Aerodynamic design optimization on unstructured grids with a continuous adjoint formulation*, Tech. Rep. 97–9, ICASE, NASA Langley Research Center, Hampton VA 23681–0001, 1997.

[3] R. J. BALLING AND J. SOBIESZCZANSKI-SOBIESKI, *Optimization of coupled systems: A critical overview of approaches*, Tech. Rep. 94–100, ICASE, NASA Langley Research Center, Hampton VA 23681–0001, 1994.

[4] J. T. BETTS, W. P. HUFFMAN, AND D. P. YOUNG, *An investigation of algorithm performance in aerodynamic design optimization*, Tech. Rep. BCSTECH–94–061, Boeing Computer Services, P.O. Box 24346, M/S 7L-68, Seattle, WA 98124-0346, 1994.

[5] J. BORGGAARD AND J. BURNS, *A PDE sensitivity equation method for optimal aerodynamic design*, Journal of Computational Physics, (To appear).

[6] E. M. CLIFF, M. HEINKENSCHLOSS, AND A. SHENOY, *An optimal control problem for flows with discontinuities*, Journal of Optimization Theory and Applications, 94 (1997). To appear in August 1997.

[7] E. J. CRAMER, J. E. DENNIS, JR., P. D. FRANK, R. M. LEWIS, AND G. R. SHUBIN, *On alternative problem formulations for multidisciplinary design optimization*, SIAM J. Optim., 4 (1994), pp. 754–776.

[8] A. DADONE AND B. GROSSMAN, *Surface Boundary Conditions for the Numerical Solution of the Euler Equations*, AIAA Journal, 32 (1994), pp. 285–293.

[9] J. E. DENNIS, M. HEINKENSCHLOSS, AND L. N. VICENTE, *Trust–region interior–point algorithms for a class of nonlinear programming problems*, Tech. Rep. TR94–45, Department of Computational and Applied Mathematics, Rice University, Houston, TX 77005–1892, 1994.

[10] A. EBERLE, A. RIZZI, AND E. H. HIRSCHEL, *Numerical Solutions of the Euler Equations for Steady Flow Problems*, vol. 34 of Notes on Numerical Fluid Mechanics, Vieweg, Wiesbaden, 1992.

[11] P. D. FRANK AND G. Y. SHUBIN, *A comparison of optimization–based approaches for a model computational aerodynamics design problem*, Journal of Computational Physics, 98 (1992), pp. 74–89.

[12] O. GHATTAS AND J.-H. BARK, *Optimal control of two– and three–dimensional Navier–Stokes flow*, Journal of Computational Physics, (1997). To appear.

[13] M. HEINKENSCHLOSS, *Projected sequential quadratic programming methods*, SIAM J. Optimization, 6 (1996), pp. 373–417.

[14] M. HEINKENSCHLOSS AND L. N. VICENTE, *Analysis of inexact trust–region interior–point SQP algorithms*, Tech. Rep. TR95–18, Department of Computational and Applied Mathematics, Rice University, Houston, TX 77005–1892, 1995.

[15] ——, *TRICE: A package of trust–region interior–point algorithms for the solution of optimal control and engineering design problems. User's guide*, tech. rep., Department of Computational and Applied Mathematics, Rice University, Houston, TX 77005–1892, 1996. Available electronically from the URL `http://www.caam.rice.edu/~trice`.

[16] C. HIRSCH, *Numerical Computation of Internal and External Flows*, vol. 2: Computational Methods for Inviscid and Viscous Flows, John Wiley & Sons, West Sussex, 1991.

[17] ——, *Numerical Computation of Internal and External Flows*, vol. 1: Fundamentals of Numerical Discretization, John Wiley & Sons, West Sussex, 1991.

[18] A. IOLLO, G. KURUVILA, AND S. TA'ASAN, *Pseudo–time method for optimal shape design using the Euler equations*, Tech. Rep. 95–59, ICASE, NASA Langley Research Center, Hampton VA 23681–0001, 1995.

[19] A. IOLLO AND M. D. SALAS, *Optimum transonic airfoils based on the Euler equations*, Tech. Rep. 96–76, ICASE, NASA Langley Research Center, Hampton VA 23681–0001, 1996.

[20] A. IOLLO, M. D. SALAS, AND S. TA'ASAN, *Shape optimization governed by the Euler equations using an adjoint method*, Tech. Rep. 93–78, ICASE, NASA Langley Research Center, Hampton VA 23681–0001, 1993.

[21] A. JAMESON, *Airfoils admitting non-unique solutions of the Euler equations*, in Proceedings of the AIAA 22nd Fluid Dynamics Plasmadynamics and Lasers Conference, AIAA Paper 91-1625, Honolulu, HI, June 24–26 1991.

[22] ——, *Optimum aerodynamic design using CFD and control theory*, in Proceedings of the AIAA 12th Computational Fluid Dynamics Conference, AIAA Paper 95-1729, San Diego, CA, June 19–22 1995.

[23] A. JAMESON AND J. REUTHER, *Control theory based airfoil design using the Euler equations*, in Proceedings of the AIAA/USAF/NASA/ISSMO 5th Symposium on Multidisciplinary Analysis & Optimization, AIAA Paper 94–4272, Panama City Beach, FL, September 7–9 1994.

[24] L. KANTOROVICH AND G. AKILOV, *Functional Analysis in Normed Spaces*, Pergamon Press, New York, 1964.

[25] R. P. NARDUCCI, *Selected Optimization Procedures for CFD-Based Shape Design involving Shock Waves or Computational Noise*, PhD thesis, Aerospace and Ocean Engineering Department, Virgina Tech, Blacksburg, Virginia, May 1995.

[26] P. A. NEWMAN, G. J.-W. HOU, AND A. C. TAYLOR, III, *Observations regarding use of advanced analysis, sensitivity analysis, and design codes in CFD*, Tech. Rep. 96–16, ICASE, NASA Langley Research Center, Hampton VA 23681–0001, 1996.

[27] J. REUTHER, A. JAMESON, J. FARMER, L. MARTINELLI, AND D. SAUNDERS, *Aerodynamic shape optimization of complex aircraft configurations via an adjoint formulation*, in Proceedings of the AIAA 34th Aerospace Sciences Meeting and Exhibit, AIAA Paper 96–0094, Reno, NV, January 15–18 1996.

[28] A. SHENOY, E. M. CLIFF, AND M. HEINKENSCHLOSS, *Thermal–fluid control via finite–dimensional approximations*, in Proceedings of the AIAA 31st Thermophysics Conference, AIAA Paper 96–1910, New Orleans, LA, June 18–20 1996.

[29] A. R. SHENOY, *Optimization Techniques Exploiting Problem Structure: Applications to Aerodynamic Design*, PhD thesis, Aerospace and Ocean Engineering Department, Virgina Tech, Blacksburg, Virginia, April 1997.

[30] J. L. THOMAS AND M. D. SALAS, *Far–field boundary conditions for transonic lifting solutions to the Euler equations*, AIAA Journal, 24 (1986), pp. 1074–1080.

[31] G. N. VANDERPLAATS, *Approximation Concepts for Numerical Airfoil Optimization*, NASA TP 1370, NASA Ames Research Center, 1979.

[32] G. N. VANDERPLAATS, *Numerical Optimization Techniques for Engineering Design with Applications*, McGraw Hill Series in Mechanical Engineering, McGraw Hill, New York, St. Louis, San Francisco, 1984.

[33] D. P. YOUNG, W. P. HUFFMAN, M. B. BIETERMAN, R. G. MELVIN, F. T. JOHNSON, C. L. HILMES, AND A. R. DUSTO, *Issues in design optimization methodology*, Tech. Rep. BCSTECH–94–007 Rev. 1, Boeing Computer Services, P.O. Box 24346, M/S 7L-68, Seattle, WA 98124-0346, 1994.

[34] D. P. YOUNG AND D. E. KEYES, *Newton's method and design optimization*, Tech. Rep. ISSTECH–96–011, Boeing Information and Support Services, P.O. Box 24346, M/S 7L-68, Seattle, WA 98124-0346, 1996.