AIAA JOURNAL
Vol. 34, No. 2, February 1996

Infeasible Path Optimal Design Methods with Applications to Aerodynamic Shape Optimization

Carlos E. Orozco*
University of Virginia, Charlottesville, Virginia 22903-2442
and
Omar N. Ghattas†
Carnegie Mellon University, Pittsburgh, Pennsylvania 15213

We consider optimal design of physical systems described by a nonlinear boundary value problem. We propose a sequential quadratic programming method specifically tailored to the solution of this class of design problems. A particular representation of the null space of the constraint Jacobian that exploits its specific structure is employed. The resulting method avoids resolution of nonlinear behavior at the optimization iterations while keeping the size of the problem as small as that of conventional approaches. Three variants of the method are developed and discussed. These entail the solution of either two or three linear systems involving the Jacobian matrix of the discretized form of the boundary value problem. The method is used to solve aerodynamic design problems involving nonlinear transonic flow. No special provisions are made for treating discontinuities, and therefore the present implementation of the method is limited to problems with no shocks. Problems with up to 90 shape design variables are solved. Numerical results demonstrate a substantial performance improvement over conventional methods.

I. Introduction

MANY efficient methods for the solution of optimal design problems of engineered systems have been developed in the past 30 years. Many problems—especially those characterized by large-scale, nonlinear systems—nevertheless still represent a considerable computational challenge, and a number of problems remain intractable. It is clear that the analysis subproblems alone can require a tremendous amount of computing resources. Shape design of aircraft wings, for example, can require simulation of compressible flow about the wing coupled with elastic wing deformation as part of the analysis procedure. Optimal control problems as well can be equally taxing: viscous drag reduction by either shape or velocity control requires the solution of the Navier—Stokes equations as a subproblem within the optimization procedure. To render such problems tractable requires an improvement in numerical methods, as well as exploitation of emerging supercomputer systems. We present here a family of infeasible path sequential quadratic programming (SQP) methods that are specially tailored to the solution of optimal design problems in which the analysis problem requires the solution of a nonlinear algebraic system of equations. This algebraic system of equations usually results from discretization of an underlying boundary value problem (BVP).

Optimal design problems consist of an objective function reflecting design goals and a set of constraints. In the cases considered here, the objective function represents a desired response of the system (e.g., a pressure or velocity distribution) and the constraints include nonlinear algebraic equations describing system behavior (these are known as the state equations). Additional design constraints can also be incorporated. The optimization variables can be classified into two types: design variables, which describe geometry (e.g., shape, thickness), and state variables, which describe system behavior for a given design (e.g., field velocities or pressures).

A standard approach to such problems is to view the state variables as implicitly depending on the design variables. By solving the state equations for given values of the design variables, both the state variables and state equations are eliminated from the problem. Gradients of functions of interest with respect to the design variables are then computed by the chain rule. See Refs. 1 and 2 for examples of this approach in aerodynamic optimal design. We call this approach path following since the iterates follow a trajectory that satisfies the governing equations at each design iteration. Reduction in size of the problem is the major advantage of path-following methods. Another advantage is that they lead to dense constraint Jacobian and Lagrangian Hessian matrices, so that standard projected Lagrangian methods (or any other nonlinear optimization technique) can be used without a need to consider sparsity. The central disadvantage of path-following methods, however, is that they require either full solution of the nonlinear flow equations at each design iteration or a combination of a nonlinear flow analysis and a flow prediction technique.[3,4] In any case, several full nonlinear flow analyses are usually required to complete the optimization process, and this can result in interactability for complex (highly nonlinear, or large, or multidisciplinary) systems.

In contradistinction to this approach, the infeasible path method treats the discrete form of the governing equations as equality constraints of the optimization problem and the state variables as optimization variables. Since the governing equations are simply nonlinear constraints in the optimization problem, they do not have to be satisfied at intermediate optimization iterations when using a projected Lagrangian method such as SQP. Hence the name infeasible path. The governing nonlinear equations are satisfied only asymptotically as the optimum is reached, which is all that is required as far as the design problem is concerned. Thus, the infeasible path method eliminates the need to solve the state equations at each design iteration. Infeasible path methods applied to aerodynamic design were pioneered by Rizk.[5] An important characteristic of infeasible path methods is that the constraint Jacobian and Lagrangian Hessian matrices are very sparse. The main disadvantage is that these matrices are usually very large, and therefore sparsity must be exploited to yield an efficient method.[6] This can be accomplished by using a general-purpose sparse optimizer, such as MINOS.[7] Some observations about the advantages and disadvantages of using sparse optimizers as compared with standard SQP methods can be found in Ref. 6. On the other hand, it is very difficult in general to make such general purpose methods work for constraint sets that can number in the tens to hundreds of thousands, as is the case of three-dimensional problems. For such problems, it is generally accepted that one has to exploit the structure of the problem.

Presented as Paper 92-4836 at the AIAA/USAF/NASA/OAI 4th Symposium on Multidisciplinary Analysis and Optimization, Cleveland, OH, Sept. 21–23, 1992; received Feb. 7, 1995; revision received Sept. 22, 1995; accepted for publication Sept. 22, 1995. Copyright © 1995 by the American Institute of Aeronautics and Astronautics, Inc. All rights reserved.

*Assistant Professor, Department of Civil Engineering and Applied Mechanics. Member AIAA.

†Associate Professor, Computational Mechanics Laboratory, Department of Civil Engineering and Environmental Engineering. Member AIAA.

Toward this end, we propose a new SQP-based method that combines the advantages of the infeasible path approach (in particular it does not require the resolution of nonlinear behavior at any design iteration) and the advantages of the path-following one (reduced size, dense matrices). The method can be equally applied to structural, fluid, or coupled problems. The method is based on a specific representation of the null space of the constraint Jacobian that exploits its particular structure. Furthermore, only the projected Hessian matrix is updated and stored, resulting in storage requirements equal to those of a path-following method. We refer to this method as the coordinate basis infeasible path (CBIP) method.

Several variants of this method are considered in Sec. II. To demonstrate its use and examine its performance, it is applied to the solution of a shape optimization problem in aerodynamics. This problem is described in Sec. III. In Sec. IV, the performance of the variants of the method are compared with standard path-following approaches, and conclusions are drawn in Sec. V.

For an extensive discussion of the merits of alternative problem formulations, in particular for multidisciplinary optimization problems, the reader is directed to the recent exposition by Cramer et al.[8] We also note the work of Frank and Shubin,[9] who compare path-following and infeasible path formulations in the context of an optimization problem governed by one-dimensional compressible flow. Readers interested in prior work on infeasible path methods for structural optimization problems should consult the references cited in Ref. 6.

II. CBIP Method

The optimization strategy proposed here is tailored to physical systems governed by nonlinear BVPs. We assume that numerical approximation of the governing equations of the BVP give rise to large, sparse coefficient matrices, such as those produced by finite difference, finite element (FE), or finite volume methods. We limit our discussion to problems that are constrained by these equations only. Extensions to problems that include additional constraints can be obtained.[10] Alternatively, if there are only a few constraints, they can be added as penalty terms to the objective function.

Given the success of SQP methods for general problems (e.g., Ref. 11), it is natural to seek an SQP strategy tailored to design problems with discrete BVP constraints. Typically, the number of design variables is much smaller than the number of state variables, and the full Hessian is sparse, indefinite, and of the order of the total number of variables. It is therefore advantageous to seek a strategy that updates the projected Hessian matrix, which is positive definite at the optimum, and the order of the design variables. Since the constraint Jacobian has a special structure and this structure is known, it is possible to obtain a basis for its null space that exploits this structure. We begin our discussion with a canonical form of the optimization problem:

Minimize

$$f(\boldsymbol{x})$$

subject to

$$\boldsymbol{h}(\boldsymbol{x}) = 0 \tag{1}$$

$$f: \mathfrak{R}^{(n+m)} \to \mathfrak{R}, \qquad \boldsymbol{h}: \mathfrak{R}^{(n+m)} \to \mathfrak{R}^n, \qquad \boldsymbol{x} \in \mathfrak{R}^{(n+m)}$$

where f represents the design objective, and $\boldsymbol{h}$ is the residual of a system of nonlinear algebraic equations arising from discretization of the underlying BVP. The $(n+m)$ variables $\boldsymbol{x}$ consist of n state variables $\boldsymbol{u}$ and m design variables $\boldsymbol{b}$. Typically $n \gg m$, since a large number of state variables arise from discretization of the domain, but only a small number of parameters are needed to carry out the design (such as in shape optimization). The path-following approach to Eq. (1) would eliminate the constraints, thereby resulting in an unconstrained optimization problem. The penalty of this procedure is the need to solve the nonlinear system $\boldsymbol{h}(\boldsymbol{x}) = 0$ at least at some of the design iterations.

In contrast, the infeasible path approach regards Eq. (1) as a nonlinearly constrained optimization problem. To solve Eq. (1) using an infeasible path SQP approach, the following quadratic program (QP) is formed:

Minimize

$$\boldsymbol{p}_k^T \boldsymbol{g}_k + \tfrac{1}{2}\boldsymbol{p}_k^T \boldsymbol{G}_L \boldsymbol{p}_k \tag{2}$$

subject to

$$\boldsymbol{A}_k \boldsymbol{p}_k = -\boldsymbol{h}_k \tag{3}$$

where $\boldsymbol{g}_k$ is the gradient of the objective function, $\boldsymbol{G}_L$ is the Hessian matrix of the Lagrangian function, $\boldsymbol{A}_k$ is the Jacobian of $\boldsymbol{h}_k$, $\boldsymbol{p}_k$ is a search direction, and the subscript k indicates optimization iteration k.

For clarity we drop the subscript k; it is understood that all quantities depending on $\boldsymbol{x}$ are evaluated at $\boldsymbol{x}_k$. Let us decompose $\boldsymbol{p}$ into two components:

$$\boldsymbol{p} = \boldsymbol{Z}\boldsymbol{p}_z + \boldsymbol{Y}\boldsymbol{p}_y \tag{4}$$

in which $\boldsymbol{Z} \in \mathfrak{R}^{(n+m)\times m}$ is a matrix whose columns form a basis for the null space of $\boldsymbol{A}$, and $\boldsymbol{Y} \in \mathfrak{R}^{(n+m)\times n}$ is chosen so that the partitioned matrix $[\boldsymbol{Z}\ \boldsymbol{Y}]$ is nonsingular. We refer to $\boldsymbol{p}_z$ as the null space step, and to $\boldsymbol{p}_y$ as the y-space step. Often, $\boldsymbol{Y}$ is chosen so that its columns span the range space of $\boldsymbol{A}^T$.[12,13] Here we take a different approach, as it will be described next. In any case, the y-space step is completely determined by substituting Eq. (4) into Eq. (3). This results in the $n \times n$ system:

$$\boldsymbol{A}\boldsymbol{Y}\boldsymbol{p}_y = -\boldsymbol{h} \tag{5}$$

The null space move is found by substituting Eq. (4) into Eq. (2) and minimizing with respect to $\boldsymbol{p}_z$, which results in the system

$$\boldsymbol{Z}^T\boldsymbol{G}_L\boldsymbol{Z}\boldsymbol{p}_z = -\boldsymbol{Z}^T(\boldsymbol{g} + \boldsymbol{G}_L\boldsymbol{Y}\boldsymbol{p}_y) \tag{6}$$

The $m \times m$ projected Hessian matrix $\boldsymbol{Z}^T\boldsymbol{G}_L\boldsymbol{Z}$ is dense but of the order of the design variables and can be naturally approximated by a quasi-Newton update. Both the path-following and the infeasible path method require the storage of this matrix. On the other hand, the long and thin matrix $\boldsymbol{Z}^T\boldsymbol{G}_L\boldsymbol{Y}$ appearing in the infeasible path approach would increase storage requirements considerably over a path-following method. Our approach is to ignore this term; Nocedal and Overton have shown that, when this is done, the resulting algorithm exhibits two-step Q-superlinear convergence (when orthonormal bases are used for $\boldsymbol{Y}$ and $\boldsymbol{Z}$).[14] A critical step in our method is the definition of appropriate y- and null space bases. Since $\boldsymbol{A}$ is large and sparse, a standard QR factorization would be too expensive. To arrive at a better way to obtain $\boldsymbol{Z}$ and $\boldsymbol{Y}$, let us examine the structure of the constraint Jacobian, when considering the following partitioning of the optimization variables:

$$\boldsymbol{x}^T = [\boldsymbol{u}^T, \boldsymbol{b}^T] \tag{7}$$

in which $\boldsymbol{u} \in \mathfrak{R}^n$ are the state variables, and $\boldsymbol{b} \in \mathfrak{R}^m$ are the design variables. The partitioned constraint Jacobian becomes

$$\boldsymbol{A} = \left[\boldsymbol{K}, \frac{\partial \boldsymbol{h}}{\partial \boldsymbol{b}}\right] \tag{8}$$

in which we have identified the Jacobian of the discrete BVP with respect to the state variables as $\boldsymbol{K}$. Now, we can define a matrix $\boldsymbol{Z}$ whose columns are orthogonal to the rows of $\boldsymbol{A}$ as

$$\boldsymbol{Z} = \begin{bmatrix} -\boldsymbol{K}^{-1}\dfrac{\partial \boldsymbol{h}}{\partial \boldsymbol{b}} \\ \boldsymbol{I} \end{bmatrix} \tag{9}$$

The y-space basis is defined simply as

$$\boldsymbol{Y} = \begin{bmatrix} \boldsymbol{I} \\ 0 \end{bmatrix} \tag{10}$$

We refer to this choice as a coordinate basis approach.[15] Clearly, the matrix $[\boldsymbol{Z}\ \boldsymbol{Y}]$ is nonsingular provided $\boldsymbol{K}$ is nonsingular, and hence

it forms a basis for $\Re^{(n+m)}$. The invertibility of $\boldsymbol{K}$ is ensured by the well-posedness of the BVP. The resulting y-space step $\boldsymbol{p}_y$ can be obtained from Eq. (5) as

$$\boldsymbol{K}\boldsymbol{p}_y = -\boldsymbol{h} \tag{11}$$

We observe that this is simply a Newton step for the nonlinear system $\boldsymbol{h} = 0$. Now the null space move $\boldsymbol{p}_z$ can be found from

$$\boldsymbol{B}_z \boldsymbol{p}_z = -\boldsymbol{g}_z \tag{12}$$

where, using Eq. (9),

$$\boldsymbol{g}_z \equiv \boldsymbol{Z}^T \boldsymbol{g} = -\left(\frac{\partial \boldsymbol{h}}{\partial \boldsymbol{b}}\right)^T \boldsymbol{K}^{-T} \boldsymbol{g}_u + \boldsymbol{g}_b \tag{13}$$

Here $\boldsymbol{B}_z$ represents a quasi-Newton approximation to the projected Hessian (the BFGS update[12] was used in the present study), and $\boldsymbol{g}_u$ and $\boldsymbol{g}_b$ represent objective gradients with respect to the state and design variables, respectively. Note that the expression for the projected gradient (13) corresponds exactly to what would have been obtained for the gradient of the objective with respect to the design variables in a path-following method (see, for example, Ref. 16).

Finally, using Eq. (4), the moves in the state and design variables take the form

$$\boldsymbol{p}_u = -\boldsymbol{K}^{-1} \frac{\partial \boldsymbol{h}}{\partial \boldsymbol{b}} \boldsymbol{p}_z + \boldsymbol{p}_y \tag{14}$$

$$\boldsymbol{p}_b = \boldsymbol{p}_z \tag{15}$$

The formula for the update of the state variables, Eq. (14), can be interpreted as being composed of two components. The first term gives a first-order approximation of the change in state variables due to a change in design variables $\boldsymbol{p}_z$ and the second term gives the change that would occur if the design were held constant and the state variables were updated according to a Newton step.

A. Algorithms

Depending on the order in which the updates of the state and design variables are carried out, it is possible to identify several variants of the method outlined in the previous paragraphs. What follows is a summary of the main features of the different algorithms that correspond to variants of both the path-following and the infeasible path approaches. Detailed descriptions of these algorithms are given subsequently.

1) Full Path Following (PFF): This corresponds to the orthodox path-following approach. A full Newton solution of the nonlinear system of equations is performed at each optimization iteration. The initial guess for the state variables at the beginning of each analysis is taken as zero.

2) Partial Path Following (PFP): This is a variant of the path-following approach in which the Newton solver is warm started. That is, the Newton solution of the nonlinear system is initiated with the solution that corresponds to the previous design iteration.

3) Coordinate Basis Infeasible Path Standard (CBIPS): This is the infeasible path algorithm that results from the direct realization of the SQP method described earlier. It requires one stiffness matrix factorization and two sets of triangular solves per optimization iteration in the absence of a line search (i.e., when the step length α is taken as 1.0).

4) Coordinate Basis Infeasible Path Modified (CBIPM): This is a variant of algorithm CBIPS in which the most recent information about the design variables is used to evaluate the stiffness matrix and the residual of the nonlinear system. It requires two stiffness matrix factorizations and two sets of triangular solves per optimization iteration in the absence of a line search.

5) Coordinate Basis Infeasible Path Coleman-Conn (CBIPC): This algorithm results from applying our CBIP ideas to a Coleman-Conn method.[17] It requires one factorization and three triangular solves per optimization iteration in the ideal case.

1. PFF Algorithm

The steps of the PFF algorithm can be arranged as follows:

- Set $k = 0$, $\boldsymbol{B}_z^0 = \boldsymbol{I}$, and $\boldsymbol{b} = \boldsymbol{b}^0$.
- Solve $\boldsymbol{h}(\boldsymbol{u}, \boldsymbol{b}^0) = 0$ for $\boldsymbol{u}^0$, using $\boldsymbol{u} = 0$ as the initial guess.
- Solve $(\boldsymbol{K}^T)^0 \boldsymbol{\lambda}^0 = \boldsymbol{g}_u^0$.
- Find $\boldsymbol{g}_z^0 = -(\partial \boldsymbol{h}^T / \partial \boldsymbol{b})^0 \boldsymbol{\lambda}^0 + \boldsymbol{g}_b^0$.
- Set $\kappa = 0.0$ and $\alpha^0 = 1.0$.
- While $\|\boldsymbol{g}_z^k\| > \epsilon_1$ and $k <$ maxiter, do the following:
  1) Find $\boldsymbol{p}_z^k = -(\boldsymbol{B}_z^k)^{-1} \boldsymbol{g}_z^k$.
  2) Set $\alpha^k = [\alpha^k / (1 + \kappa \|\boldsymbol{g}_z^k\|)]$.
  3) Update design variables: $\boldsymbol{b}^{k+1} = \boldsymbol{b}^k + \alpha^k \boldsymbol{p}_z^k$.
  4) Solve $\boldsymbol{h}(\boldsymbol{u}, \boldsymbol{b}^{k+1}) = 0$ for $\boldsymbol{u}^{k+1}$, using $\boldsymbol{u} = 0$ as the initial guess.
  5) If $(f^k - f^{k+1}) \geq -\mu \alpha^k \boldsymbol{g}_z^k \boldsymbol{p}_z^k$, go to step 6 or else increment κ and go back to step 2 (this is one of the so-called strong Wolfe conditions; see, for example, Refs. 12 and 13).
  6) Solve $(\boldsymbol{K}^T)^{k+1} \boldsymbol{\lambda}^{k+1} = \boldsymbol{g}_u^{k+1}$.
  7) Find $\boldsymbol{g}_z^{k+1} = -(\partial \boldsymbol{h}^T / \partial \boldsymbol{b})^{k+1} \boldsymbol{\lambda}^{k+1} + \boldsymbol{g}_b^{k+1}$.
  8) Find $\boldsymbol{y}_z^k = \boldsymbol{g}_z^{k+1} - \boldsymbol{g}_z^k$.
  9) Update $\boldsymbol{B}_z^k$ using $\boldsymbol{y}_z^k$ and $\boldsymbol{p}_z^k$.
  10) Set $\boldsymbol{g}_z^k = \boldsymbol{g}_z^{k+1}$.
  11) Set $k = k + 1$.
- Endwhile.

Note that the most expensive step of this algorithm is the solution of the set of nonlinear algebraic equations in step 4. When a Newton method is used, this step normally takes on the order of 5–12 linear solves (for the problems considered in this study).

2. PFP Algorithm

The PFP algorithm is obtained by using $\boldsymbol{u}^k$ as the initial guess for the nonlinear system solve in the preceding step 4.

3. CBIPS Algorithm

The steps of the standard CBIP approach can be arranged into the following algorithm, which we refer to as CBIPS.

- Set $k = 0$, $\boldsymbol{B}_z^0 = \boldsymbol{I}$, $\boldsymbol{u} = \boldsymbol{u}^0$, and $\boldsymbol{b} = \boldsymbol{b}^0$.
- Find $\boldsymbol{h}^0 = \boldsymbol{h}(\boldsymbol{u}^0, \boldsymbol{b}^0)$.
- Solve $(\boldsymbol{K}^T)^0 \boldsymbol{\lambda}^0 = \boldsymbol{g}_u^0$.
- Find $\boldsymbol{g}_z^0 = -(\partial \boldsymbol{h}^T / \partial \boldsymbol{b})^0 \boldsymbol{\lambda}^0 + \boldsymbol{g}_b^0$.
- Set $k = 0.0$ and $\alpha^0 = 1.0$.
- While $(\|\boldsymbol{g}_z^k\| > \epsilon_1$ or $\|\boldsymbol{h}^k\| > \epsilon_2)$ and $k <$ maxiter, do the following:
  1) Find $\boldsymbol{p}_z^k = -(\boldsymbol{B}_z^k)^{-1} \boldsymbol{g}_z^k$.
  2) Set $\alpha^k = [\alpha_k / (1 + \kappa \|\boldsymbol{g}_z^k\|)]$.
  3) Find $\boldsymbol{d}^k = (\partial \boldsymbol{h}^k / \partial \boldsymbol{b})(\alpha^k \boldsymbol{p}_z^k) + \boldsymbol{h}^k$.
  4) Solve $\boldsymbol{K}^k \boldsymbol{p}_u^k = -\boldsymbol{d}^k$.
  5) Update variables: $\boldsymbol{u}^{k+1} = \boldsymbol{u}^k + \boldsymbol{p}_u^k$, and $\boldsymbol{b}^{k+1} = \boldsymbol{b}^k + \alpha^k \boldsymbol{p}_z^k$.
  6) If $(f^k - f^{k+1}) \geq -\mu \alpha^k \boldsymbol{g}_z^k \boldsymbol{p}_z^k$, go to step 7 else increment κ and go back to step 2.
  7) Solve $(\boldsymbol{K}^T)^{k+1} \boldsymbol{\lambda}^{k+1} = \boldsymbol{g}_u^{k+1}$.
  8) Find $\boldsymbol{g}_z^{k+1} = -(\partial \boldsymbol{h}^T / \partial \boldsymbol{b})^{k+1} \boldsymbol{\lambda}^{k+1} + \boldsymbol{g}_b^{k+1}$.
  9) Find $\boldsymbol{y}_z^k = \boldsymbol{g}_z^{k+1} - \boldsymbol{g}_z^k$.
  10) Update $\boldsymbol{B}_z^k$ using $\boldsymbol{y}_z^k$ and $\boldsymbol{p}_z^k$.
  11) Set $\boldsymbol{g}_z^k = \boldsymbol{g}_z^{k+1}$.
  12) Set $k = k + 1$.
- Endwhile.

Note here that this algorithm requires the solution of just two linear systems of equations having the same coefficient matrix. This is in contrast to the PFF (or PFP) algorithms that normally require on the order of 5–12 (or 4–5) linear solves. Note also that steps 2–6 constitute a line search in the design space. When $\boldsymbol{K}$ is symmetric (as in the present case), it is necessary to factor only once and perform two triangular solves. Tests have shown that this CBIP method is more efficient than a path-following method even for problems that are mildly nonlinear (i.e., that require 4–6 Newton steps per iteration). A CBIP algorithm with improved performance is obtained if the CBIPS is modified in such a way that it yields exactly the same sequence of iterates as the path-following one, for linear BVPs (i.e., when $\boldsymbol{K}$ is not a function of $\boldsymbol{u}$). Two modifications are made: first, the quantity $\boldsymbol{d}^k$ in step 3 is replaced by $\boldsymbol{d}^k = \boldsymbol{h}(\boldsymbol{b}^{k+1}, \boldsymbol{u}^k)$ [note that $(\partial \boldsymbol{h} / \partial \boldsymbol{b})^k (\alpha^k \boldsymbol{p}_z^k) + \boldsymbol{h}^k$ is a first-order approximation of $\boldsymbol{h}(\boldsymbol{b}^{k+1}, \boldsymbol{u}^k)$], and second, $\boldsymbol{K}$ in step 4 is evaluated at the newest value of $\boldsymbol{b}$, i.e., at

$\boldsymbol{b}^{k+1}$. With these two modifications, we get an improved algorithm, which we refer to as the modified CBIP algorithm (CBIPM).

4. CBIPM Algorithm

Following the foregoing discussion, algorithm CBIPM can be obtained from algorithm CBIPS by replacing steps 3, 4, and 5 with the following:

3) Update the design variables: $\boldsymbol{b}^{k+1} = \boldsymbol{b}^k + \alpha^k \boldsymbol{p}_z^k$.
4) Solve $\boldsymbol{K}(\boldsymbol{u}^k, \boldsymbol{b}^{k+1})\boldsymbol{p}_u^k = -\boldsymbol{h}(\boldsymbol{u}^k, \boldsymbol{b}^{k+1})$.
5) Update state variables: $\boldsymbol{u}^{k+1} = \boldsymbol{u}^k + \boldsymbol{p}_u^k$.

5. CBIPC Algorithm

The CBIPC algorithm can be obtained from the CBIPM as follows:

- Replace step 4 with the following:
  4) Solve $\boldsymbol{K}^k \boldsymbol{p}_{uz}^k = -(\partial \boldsymbol{h}^k / \partial \boldsymbol{b})\alpha^k \boldsymbol{p}_z^k$.
- Add step 5, as follows:
  5) Solve $\boldsymbol{K}^k \boldsymbol{p}_{uy}^k = -\boldsymbol{h}(\boldsymbol{u}^k + \boldsymbol{p}_{uz}^k, \boldsymbol{b}^{k+1})$.
- Replace step 5 with the following:
  6) Update states variables: $\boldsymbol{u}^{k+1} = \boldsymbol{u}^k + (\boldsymbol{p}_{uz}^k + \boldsymbol{p}_{uy}^k)$.

The remaining steps are as for the CBIPM algorithm. The key difference of the Coleman–Conn algorithm with respect to the other two infeasible path algorithms is its requirement of an extra constraint evaluation that results in an extra triangular solve.

III. Aerodynamic Design Problem

A design problem of a system governed by a nonlinear partial differential equation is defined in this section. The problem consists of finding the shape of the airfoil that exhibits a prescribed (or target) pressure distribution. The flow around the airfoil is modeled by the nonlinear full-potential equations.

Let Ω represent the domain of definition, Γ_s the surface of the airfoil, Γ_w a split boundary intended to model the wake, Γ_∞ the far-field boundary, $\boldsymbol{u}_\infty$ the freestream velocity, and α the angle of attack of the airfoil.

The analysis problem seeks the pressure distribution over Γ_s. The equation of continuity $\nabla \cdot \{\rho(\boldsymbol{u})\boldsymbol{u}\} = 0$, together with appropriate boundary conditions, gives rise to the following BVP:

$$\begin{aligned} \nabla \cdot \{\tilde{\rho} \nabla \phi\} &= 0 \quad \text{in} \quad \Omega \\ \nabla \phi \cdot \boldsymbol{n} &= 0 \quad \text{on} \quad \Gamma_s \\ \nabla \phi \cdot \boldsymbol{n} &= \boldsymbol{u}_\infty \cdot \boldsymbol{n} \quad \text{on} \quad \Gamma_\infty \end{aligned} \tag{16}$$

in which $\tilde{\rho}$ is an upwind approximation of the density, and ϕ is the velocity potential.

The corresponding shape optimization problem is to find the shape that induces the desired pressure distribution and can be stated as follows:

Minimize

$$f(\Omega_h, \Phi) = \int_{\Gamma_s} [p(\Phi) - p^*]^2 \, d\Gamma + \delta[p^u(\Omega, \Phi) - p^l(\Omega, \Phi)]^2 \tag{17}$$

subject to

$$\begin{aligned} &\int_{\Omega_h} \Bigg[\{1 - [(\gamma - 1)/2](\Phi^T \boldsymbol{Q} \Phi)\}^{1/(\gamma-1)} \\ &\quad - \tilde{h}_s \left(\frac{\{[(\gamma+1)/2]\Phi^T \boldsymbol{Q} \Phi - 1\}^+}{[\Phi^T \boldsymbol{Q} \Phi]^{\frac{3}{2}}} \Phi^T \boldsymbol{Q} \rho \right) \Bigg] \boldsymbol{Q} \Phi \, d\Omega \\ &= \int_{\Gamma_{h\infty}} \boldsymbol{N}(\rho_\infty \boldsymbol{u}_\infty) \cdot \boldsymbol{n} \, d\Gamma \end{aligned} \tag{18}$$

where Φ and ρ are discretized versions of ϕ and ρ, $\boldsymbol{Q}$ is a matrix of products of interpolation functions, γ is the ratio of specific heats, $p(\Phi)$ is the predicted pressure on the surface of the airfoil, p^* is the prescribed pressure, and $\tilde{h}_s$ is a mesh size dependent arc length in the streamline direction. The quantity between the brackets is just a discretized version of the upwind approximation of the density [of $\tilde{\rho}$ in Eq. (16)].[18] The constraint (18) corresponds to a FE Galerkin approximation of the BVP. The second term in the objective is a penalty term to enforce the Kutta condition at the trailing edge. The boundary conditions are implied in Eq. (18). This optimization problem is of the general form of Eq. (1).

IV. Numerical Results

Next we compare the performance of the five algorithms for the aerodynamic design problem defined earlier. We do this for NACA airfoils. In the first set of examples, the design variables are the basic parameters of the NACA family of airfoils, namely, the maximum thickness τ, the position of the maximum camber p, and the maximum camber ϵ. In the second set of examples, the airfoil is parameterized using cubic splines (Fig. 1).

A. Performance with Increasing Mach Number

The numerical results of this section are intended to illustrate the behavior of the algorithms when the nonlinearity of the state equations increases. To this end, the total number of iterations, total CPU time, and number of linear solves were recorded for a series of problems with increasing Mach numbers. Since no special provisions are made to allow for discontinuities either at the optimization or at the analysis level, the results are limited to problems with no shocks.

These problems were run on an SGI Indigo2 workstation. The linear systems involving $\boldsymbol{K}$ were solved using ma28.f, the general direct sparse solver available from the Harwell library. The target pressure distribution was taken as that of a NACA 2412 airfoil at $\alpha = 1.0$ and Mach numbers from 0.40 to 0.68. The discretization used 3280 linear triangular FEs and 1743 nodes. The corresponding mesh with the target airfoil is shown in Fig. 2. Many problems for different initial guesses were run. Table 1 summarizes the results for a problem with $\tau = 0.09$, $p = 0.34$, and $\epsilon = 0.005$, as the initial guess. The target (optimum) values were $\tau = 0.12$, $p = 0.40$, and $\epsilon = 0.02$ (NACA 2412).

Convergence criteria for all algorithms were established in terms of the norm of the projected gradient and the norm of the residual of the state equations. The tolerances were set at $\epsilon_1 = 1.0 \times 10^{-3}$ for $\|\boldsymbol{g}_z\|$ and $\epsilon_2 = 1.0 \times 10^{-7}$ for $\|\boldsymbol{h}\|$. It can be observed that the pertinent numbers in Table 1 are below these values.

Total CPU times are plotted against Mach numbers in Fig. 3. It can be seen that the path-following algorithms require the most CPU

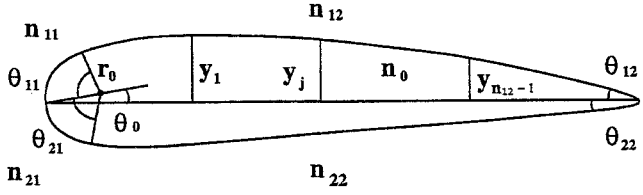

Fig. 1 Parameterization of airfoil using cubic splines.

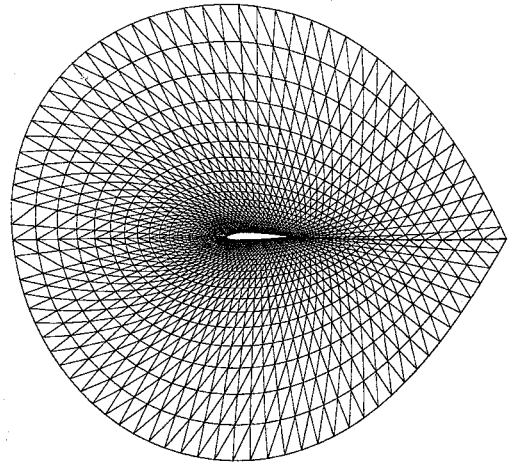

Fig. 2 Typical discretized two-dimensional domain with airfoil.

Table 1 Performance with increasing Mach number: angle of attack = 1.0 deg, discretization: 3280 FE, 1743 flow variables, and 3 design variables

M_∞	Method	f	$\|g_z\|$	$\|h\|$	$\|p_z\|$	CPU	Iter.	Linear solves	CPU/ iter.	Solves/ iter.
0.40	PFF	0.1970D-10	0.1915D-04	0.2758D-06	0.1804D-03	168	10	50	16.8	5.0
	PFP	0.1453D-10	0.1912D-04	0.1342D-08	0.1804D-03	141	10	39	14.1	3.9
	CBIPS	0.4803D-11	0.1282D-04	0.3096D-08	0.1776D-04	121	16	35	7.6	2.2
	CBIPC	0.1104D-10	0.2046D-04	0.4046D-06	0.1173D-03	108	14	46	7.7	3.3
	CBIPM	0.6046D-13	0.1947D-05	0.8479D-09	0.2629D-05	150	15	33	10.0	2.2
0.45	PFF	0.1460D-10	0.1909D-04	0.3308D-11	0.1766D-03	191	10	63	19.1	6.3
	PFP	0.1463D-10	0.1903D-04	0.3198D-08	0.1761D-03	149	10	41	14.9	4.1
	CBIPS	0.1825D-11	0.1048D-04	0.4344D-09	0.7888D-05	128	17	37	7.5	2.2
	CBIPC	0.1198D-09	0.4282D-04	0.7919D-08	0.2433D-03	108	14	46	7.7	3.3
	CBIPM	0.5385D-10	0.4308D-04	0.6857D-07	0.1719D-03	137	14	31	9.8	2.2
0.50	PFF	0.1612D-10	0.1969D-04	0.1077D-09	0.1767D-03	195	10	63	19.5	6.3
	PFP	0.1615D-10	0.1962D-04	0.7051D-08	0.1766D-03	146	10	41	14.6	4.1
	CBIPS	0.3302D-09	0.6727D-04	0.7424D-06	0.6304D-04	121	16	35	7.6	2.2
	CBIPC	0.1420D-09	0.4504D-04	0.8924D-08	0.2499D-03	115	14	46	8.2	3.3
	CBIPM	0.7577D-10	0.4140D-04	0.1077D-06	0.1325D-03	128	14	31	9.9	2.2
0.55	PFF	0.1936D-10	0.2071D-04	0.3037D-08	0.1792D-03	197	10	63	19.7	6.3
	PFP	0.1922D-10	0.2075D-04	0.1492D-07	0.1793D-03	150	10	42	15.0	4.2
	CBIPS	0.3236D-09	0.1024D-03	0.9781D-06	0.3964D-04	115	15	33	7.7	2.2
	CBIPC	0.4755D-12	0.3831D-05	0.4619D-06	0.5131D-04	93	12	40	7.8	3.3
	CBIPM	0.4781D-11	0.9760D-05	0.1095D-08	0.1180D-04	142	14	31	10.1	2.2
0.60	PFF	0.2887D-10	0.1815D-04	0.8323D-07	0.1512D-03	202	10	68	20.2	6.8
	PFP	0.1494D-10	0.1819D-04	0.3091D-07	0.1512D-03	164	10	48	16.4	4.8
	CBIPS	0.1673D-10	0.6042D-05	0.8921D-08	0.3269D-04	114	15	33	7.6	2.2
	CBIPC	0.1653D-10	0.3472D-04	0.4643D-06	0.5783D-04	86	11	37	7.8	3.4
	CBIPM	0.3988D-11	0.1416D-04	0.1131D-08	0.1645D-04	146	15	33	9.7	2.2
0.65	PFF	0.8551D-10	0.6766D-04	0.5362D-08	0.9660D-04	196	9	66	21.8	7.3
	PFP	0.8847D-10	0.6868D-04	0.1450D-06	0.9793D-04	140	9	40	15.6	4.4
	CBIPS	0.9956D-11	0.2775D-04	0.4217D-08	0.2775D-05	136	18	40	7.6	2.2
	CBIPC	0.2032D-10	0.1585D-04	0.1170D-07	0.1404D-03	100	13	42	7.7	3.2
	CBIPM	0.1976D-08	0.4454D-03	0.4012D-06	0.5758D-04	129	13	29	9.9	2.2
0.68	PFF	0.1161D-08	0.8111D-04	0.6108D-06	0.1153D-03	311	9	113	34.6	12.6
	PFP	0.8814D-09	0.4554D-03	0.4958D-06	0.3817D-05	220	9	72	24.4	8.0
	CBIPS	0.1716D-08	0.8743D-03	0.4002D-06	0.1509D-04	190	24	63	7.9	2.6
	CBIPC	0.2581D-09	0.2529D-03	0.2385D-06	0.3016D-05	140	18	61	7.8	3.4
	CBIPM	0.1774D-08	0.8203D-04	0.9824D-06	0.7980D-04	179	17	41	10.5	2.4

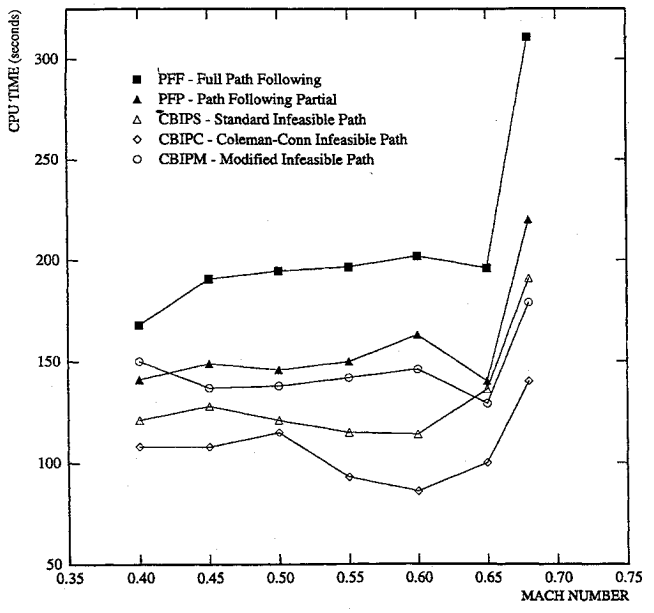

Fig. 3 CPU time vs Mach number. Angle of attack = 1.0 deg; three design variables. Discretization: 3280 FE, 1743 flow variables.

time. The path-following algorithms also require greater CPU time for higher Mach numbers, due to the greater degree of nonlinearity. The number of optimization iterations is least for the path-following algorithms (see Table 1). This is because additional optimization iterations are required to converge the flow equations for the infeasible path methods. In a sense, part of what is saved by avoiding nonlinear solutions is lost due to more optimization iterations. Nevertheless, the CBIP methods result in significant computational savings, and the savings tend to increase as the nonlinearity and the size of the analysis problem increases.

The number of optimization iterations is about the same for CBIPM and CBIPC, with CBIPC tending to take slightly fewer iterations than CBIPM. Our experience shows that CBIPS experiences some convergence difficulties and (depending on the initial guess) tends to oscillate about the optimum, requiring in general more iterations than CBIPC and CBIPM. This type of oscillating behavior has been observed also for the PFP algorithm. On average, 4.8 linear solves are required by PFP at each iteration, and 7.2 linear solves by PFF. In contrast, only 2.2 linear solves are performed by either CBIPS or CBIPM, and 3.3 by CBIPC (see Table 1). The number of linear solves is not exactly equal to the theoretical value for the CBIP algorithms because additional linear solves are required to complete some of the line searches. The linear solves reported in Table 1 for CBIPS and CBIPC include some triangular solves. Approximately one-half of the linear solves reported for CBIPS and two-thirds of the ones reported for CBIPC are actually triangular solves. This results in less total CPU times for CBIPC and CBIPS (with respect to CBIPM) as can be observed in Table 1 and in Fig. 3.

If iterative methods are used to solve the linear systems involving $\boldsymbol{K}$, it is not possible to take advantage of having the same coefficient matrix with different right-hand sides. In such a case, the three variants of the infeasible path methods will require at least two linear solves for CBIPM and CBIPS and three linear solves for CBIPC. Since the remaining work per iteration is practically identical for all algorithms, and our experience indicates that CBIPM requires in general fewer iterations than CBIPS, it can be concluded that when iterative methods are used to solve the linear systems, CBIPM will be more efficient than CBIPC and CBIPM. This can be inferred from Fig. 4 where it is seen that CBIPM takes the least number of linear solves. This has also been confirmed for the case of a massively parallel implementation of the algorithms.[19,10]

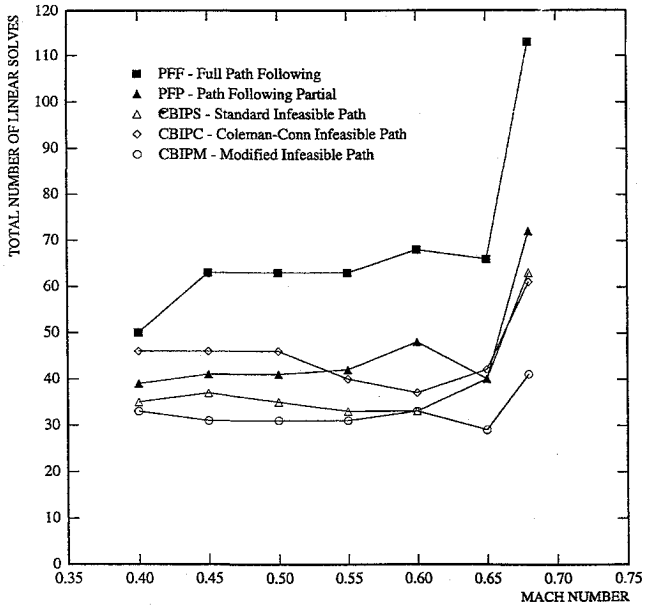

Fig. 4 Total number of linear solves vs Mach number. Angle of attack = 1.0 deg; three design variables. Discretization: 3280 FE, 1743 flow variables.

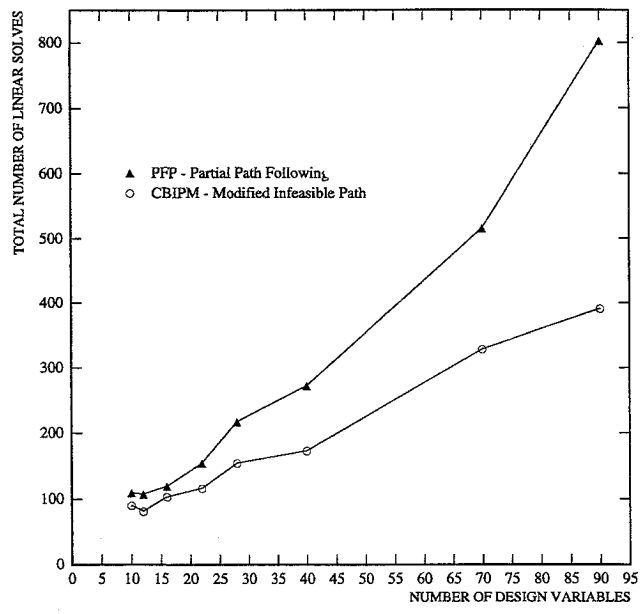

Fig. 5 Total number of linear solves vs number of design variables. $M_\infty = 0.6$, angle of attack = 1.0 deg. Discretization: 3952 FE, 2079 flow variables.

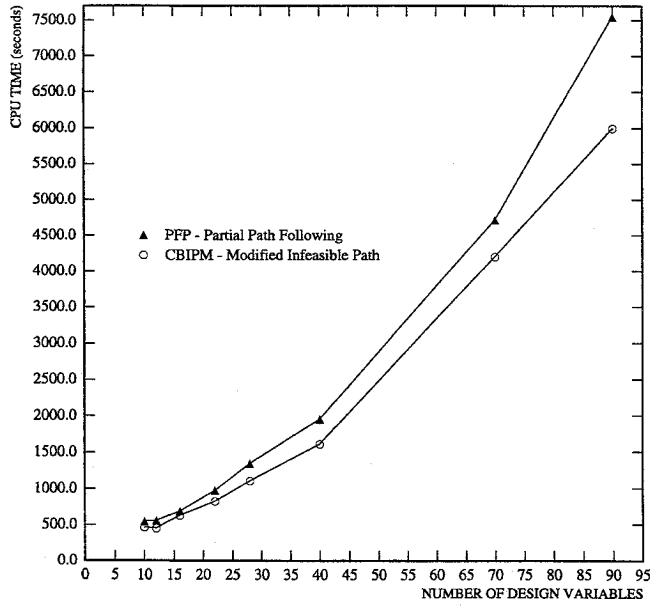

Fig. 6 Total CPU time vs number of design variables. $M_\infty = 0.6$, angle of attack = 1.0 deg. Discretization: 3952 FE, 2079 flow variables.

B. Sensitivity Analysis

All algorithms require the calculation of the projected gradient (13). This is done by using the adjoint method of sensitivity analysis[16] (see, e.g., steps 7 and 8 of the CBIPS algorithm). From the optimization point of view, step 7 can also be interpreted as a way to find a first-order approximation of the Lagrange multipliers. The quantity $\partial \boldsymbol{h}/\partial \boldsymbol{b}$ was estimated using central differences (this of course involves remeshing twice for each design variable). The quantity $\boldsymbol{g}_u$ was also estimated using central differences.

C. Performance with Increasing Number of Design Variables

To allow for more than three design variables (as when using the NACA parameters for modeling the airfoil) and to simulate a real design situation, a second parameterization based on cubic splines was used to describe the shape of the airfoil (see Fig. 1). The leading edge of the airfoil is modeled with a circular arc of radius r_0. This arc extends θ_{11} and θ_{21} above and below a line that is inclined θ_0 with respect to the horizontal axis. These quantities can have any positive value and constitute design variables. For the purpose of the FE discretization the arc portions corresponding to θ_{11} and θ_{21} are divided into n_{11} and n_{21} subintervals that correspond to edges of triangular FE. The remaining upper portion of the airfoil is divided into n_{12} intervals. For the FE discretization each of these intervals is divided into n_0 subintervals that again correspond to FE; $(n_{12}-1)y$ coordinates are then used to fit a cubic spline between the end of the arc and the trailing edge. These y coordinates are also taken as design variables. The lower portion of the airfoil is modeled in an analogous way. Angles θ_{12} and θ_{22} at the trailing edge complete the list of design variables. The total number of design variables with this model is thus $n_{12}+n_{22}+4$. This parameterization proved to be very flexible and allowed a very accurate modeling of airfoils.

As before, the problems presented in this section were run on an SGI Indigo2 workstation. All results, with the exception of the 90-variable case, correspond to an airfoil design problem with $M_\infty = 0.6, \alpha = 1.0$, 3952 FE, and 2079 nodes. The 90-variable case corresponds to a design problem with flow about a unit cylinder with $M_\infty = 0.37$, 4416 FE, and 2325 nodes.

We compare here the performance of the CBIPM algorithm with that of the PFP. The shape parameterization described earlier allows an increase in the number of design variables m while maintaining the number of FE and hence the number of state variables n, fixed.

Results are summarized in Table 2. The total CPU times and the total number of linear solves are plotted against the number of design variables in Figs. 5 and 6. CPU times and number of solves increase with the number of design variables because the number of optimization iterations normally increases with the dimension of the design space. For the set of problems presented in this section, the CBIPM algorithm was started using a fully converged solution of the flow equations. As a consequence of this, the number of optimization iterations is basically the same for the CBIPM and PFP algorithms. The number of linear solves and hence the cost per iteration, however, are much smaller for the CBIPM algorithm. For the 90-variable case, the number of linear solves decreases from 802 for PFP to 390 for CBIPM. The corresponding decrease in CPU time is of the order of 20% (see Table 2 and Fig. 6).

D. Accuracy

The gradient tolerance of 1.0×10^{-3} ensures that the norm of the change in the design variables ($\|\boldsymbol{p}_z\|$) at optimality is also approximately of the order of 1.0×10^{-3}. For a problem with 40 design variables this means that the optimum values of the design variables are accurate up to the third decimal digit (which should be enough for most engineering applications). On the other hand, for a problem with 2,300 state variables, the tolerance of 1.0×10^{-7} for the

Table 2 Performance with increasing number of design variables; $M_\infty = 0.6$ angle of attack = 1.0 deg, discretization: 3952 FE and 2079 flow variables

No. of design variables	Method	f	$\|g_z\|$	$\|h\|$	$\|p_z\|$	CPU	Iter.	Linear solves	CPU/ iter.	Solves/ iter.
10	PFP	0.2591D-02	0.2094D-03	0.3875D-08	0.8625D-04	540	37	109	14.6	2.9
	CBIPM	0.2591D-02	0.2340D-03	0.2709D-06	0.6388D-04	461	37	90	12.5	2.4
12	PFP	0.1355D-02	0.9363D-04	0.3106D-06	0.1683D-02	551	36	107	15.3	3.0
	CBIPM	0.1355D-02	0.1181D-03	0.2315D-11	0.7299D-04	447	36	81	12.4	2.3
16	PFP	0.3927D-03	0.7292D-04	0.2348D-06	0.1889D-02	683	43	119	15.9	2.8
	CBIPM	0.3927D-03	0.9542D-04	0.7989D-07	0.1060D-02	624	46	103	13.6	2.2
22	PFP	0.1517D-03	0.4687D-04	0.5367D-07	0.1214D-02	968	53	154	18.3	2.9
	CBIPM	0.1517D-03	0.2058D-04	0.2033D-07	0.8713D-03	818	53	116	15.4	2.2
28	PFP	0.9857D-04	0.5060D-04	0.3016D-06	0.3328D-02	1341	56	217	23.9	3.9
	CBIPM	0.9856D-04	0.4694D-04	0.7038D-07	0.2081D-02	1101	58	154	19.0	2.7
40	PFP	0.5041D-04	0.4610D-04	0.2347D-07	0.7250D-03	1948	69	272	28.2	3.9
	CBIPM	0.5040D-04	0.6951D-04	0.6832D-07	0.1206D-02	1607	72	173	22.3	2.4
70	PFP	0.6315D-04	0.8195D-04	0.1047D-08	0.2607D-03	4711	109	515	43.2	4.7
	CBIPM	0.6313D-04	0.1210D-03	0.1097D-08	0.2438D-04	4197	115	328	36.5	2.9
90	PFP	0.2349D-07	0.5848D-03	0.8453D-08	0.4778D-04	7545	123	802	61.3	6.5
	CBIPM	0.3388D-07	0.7389D-03	0.1198D-03	0.5959D-04	5996	129	390	46.5	3.0

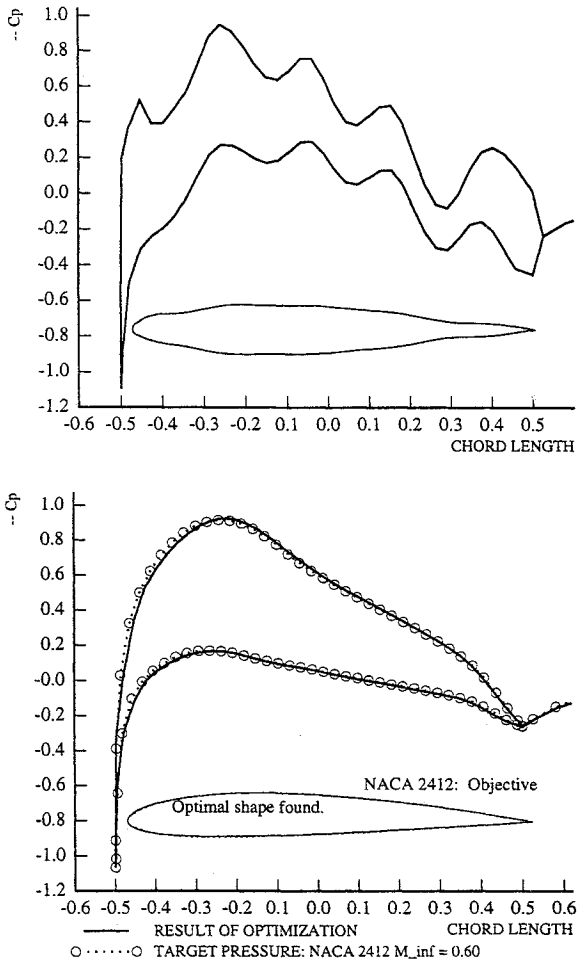

Fig. 7 Arbitrary initial shape with corresponding pressure distribution. Splines model with 22 shape design variables. $M_\infty = 0.6$, $\alpha = 1.0$ deg. Objective: NACA 2412.

residual of the state equations means that the residuals are zero up to the eighth significant digit.

E. Line Search

The step lengths were limited to avoid invalid meshes and/or undefined values of K and h and to ensure a sufficient decrease in the objective function (see steps 5 and 6 of the algorithms). For the problems parameterized with the splines model, it was also necessary to limit the change of each design variable to 40% in any given optimization iteration.

F. Sample Design

A typical design example is presented in Fig. 7. The objective is the NACA 2412 at $M_\infty = 0.6$ and $\alpha = 1.0$. The figure illustrates the arbitrary initial shape, the corresponding initial pressure distribution, the target pressure distribution and airfoil, and the optima found. This case corresponds to the splines model with $n_{12} = n_{22} = 9$ (22 shape variables). It is seen that the optimum pressures and shape practically coincide with the target ones.

V. Conclusions

We have presented an SQP-based infeasible path method for the optimal design of systems governed by nonlinear BVP. In particular, we have considered shape optimization of airfoils in compressible transonic flow with no shocks. The infeasible path methods presented do not require full resolution of the flow equations at the optimization iterations. This is accomplished by including the governing equations as equality constraints in the optimization problem. In addition, the methods exhibit the advantages of path-following methods in the sense that they have the same storage requirements and that they require the calculation of the same quantities. This is done by means of a null and y-space decomposition of the constraint Jacobian that exploits the special structure of the discretized BVP. The resulting methods require the solution of just two or three linear systems at each optimization iteration. The coefficient matrix of these systems is just the Jacobian matrix of the flow equations, thereby enabling the methods to leverage efficient flow solvers. Examples demonstrate a savings in computational effort of about 20% for nonlinear aerodynamic design problems with around 4400 FE and 90 shape design variables.

Acknowledgments

This work was partially supported by the Engineering Design Research Center, a National Science Foundation (NSF) Engineering Research Center at Carnegie Mellon University, and by NSF Grant DDM-9114678. We thank Larry T. Biegler of the Department of Chemical Engineering at Carnegie Mellon University for several useful discussions. The authors highly appreciate the thoroughness of the reviewers' comments. They greatly contributed to the improvement of the overall quality of the paper.

References

[1]Burgreen, G. W., Baysal, O., and Eleshaky, M. E., "Improving the Efficiency of Aerodynamic Shape Optimization," *AIAA Journal*, Vol. 32, No. 1, 1994, pp. 69–76.

[2]Burgreen, G. W., and Baysal, O., "Aerodynamic Shape Optimization Using Preconditioned Conjugate Gradient Methods," *AIAA Journal*, Vol. 32, No. 11, 1994, pp. 2145–2152.

[3]Baysal, O., and Eleshaky, M., "Aerodynamic Design Optimization Using Sensitivity Analysis and Computational Fluid Dynamics," *AIAA Journal*, Vol. 30, No. 3, 1992, pp. 718–725.

[4]Baysal, O., Eleshaky, M., and Burgreen, G., "Aerodynamic Shape Optimization Using Sensitivity Analysis on Third-Order Euler Equations," *AIAA Journal*, Vol. 30, No. 6, 1992, pp. 953–961.

[5]Rizk, M. H., "The Single-Cycle Scheme: A New Approach to Numerical Optimization," *AIAA Journal*, Vol. 21, No. 12, 1983, pp. 1640–1647.

[6]Orozco, C. E., and Ghattas, O. N., "Sparse Approach to Simultaneous Analysis and Design of Geometrically Nonlinear Structures," *AIAA Journal*, Vol. 30, No. 7, 1992, pp. 1877–1885.

[7]Murtagh, B. A., and Saunders, M. A., "MINOS 5.1 User's Guide," Dept. of Operations Research, Stanford Univ., TR SOL 83-20R, Stanford, CA, Jan. 1987.

[8]Cramer, E. J., Dennis, J. E., Frank, P. D., Lewis, R. M., and Shubin, G. R., "Problem Formulation for Multidisciplinary Optimization," *SIAM Journal on Optimization*, Vol. 4, No. 4, 1994, pp. 754–776.

[9]Frank, P. D., and Shubin, G. R., "A Comparison of Optimization-Based Approaches for a Model Computational Aerodynamic Design Problem," *Journal of Computational Physics*, Vol. 98, No. 1, 1992, pp. 74–89.

[10]Orozco, C. E., "Large-Scale Shape Optimization: Numerical Methods, parallel Algorithms, and Applications to Aerodynamic Design," Ph.D. Thesis, Dept. of Civil Engineering, Carnegie Mellon Univ., Pittsburgh, PA, 1993.

[11]Schittkowski, K., "NLPQL: A FORTRAN Subroutine for Solving Constrained Nonlinear Programming Problems," *Annals of Operations Research*, Vol. 5, No. 6, 1985, pp. 485–500.

[12]Gill, P. E., Murray, W., and Wright, M. H., *Practical Optimization*, Academic, New York, 1981.

[13]Fletcher, R., *Practical Methods of Optimization*, 2nd ed., Wiley, New York, 1987.

[14]Nocedal, J., and Overton, M. L., "Projected Hessian Updating Algorithms for Nonlinearly Constrained Optimization," *SIAM Journal on Numerical Analysis*, Vol. 22, No. 5, 1985, pp. 821–850.

[15]Biegler, L. T., Nocedal, J., and Schmid, C., "A Reduced Hessian Method for Large Scale Constrained Optimization," *SIAM Journal on Optimization*, Vol. 4, No. 2, 1995, pp. 314–347.

[16]Haftka, R. T., Gürdal, Z., and Kamat, M. P., *Elements of Structural Optimization*, Kluwer Academic, Boston, 1990.

[17]Coleman, T. F., and Conn, A. R., "Nonlinear Programming via an Exact Penalty Function: Asymptotic Analysis," *Mathematical Programming*, Vol. 24, No. 2, 1982, pp. 123–136.

[18]Bristeau, M. O., Glowinski, R., Periaux, J., Perrier, P., Pironneau, O., and Poirier, G., "Transonic Flow Simulations by Finite Element and Least-Square Methods," *Finite Elements in Fluids*, edited by R. H., Gallagher, D. H. Norries, J. T. Oden, and O. C. Zienkiewicz, Vol. 4, Wiley, New York 1982, pp. 453–481.

[19]Orozco, C. E., and Ghattas, O. N., "Massively Parallel Aerodynamic Shape Optimization," *Computing Systems in Engineering*, Vol. 3, No. 14, 1992, pp. 311–320.