JOURNAL OF COMPUTATIONAL PHYSICS **136,** 231–244 (1997)
ARTICLE NO. CP975744

Optimal Control of Two- and Three-Dimensional Incompressible Navier–Stokes Flows[1]

Omar Ghattas* and Jai-Hyeong Bark†

**Computational Mechanics Laboratory, Department of Civil and Environmental Engineering, Carnegie Mellon University, Pittsburgh, Pennsylvania 15213; and †Department of Architectural Engineering, Mokwon University, Taejon, South Korea*
E-mail: Omar.Ghattas@cs.cmu.edu

Received April 2, 1996; revised March 3, 1997

The focus of this work is on the development of large-scale numerical optimization methods for optimal control of steady incompressible Navier–Stokes flows. The control is affected by the suction or injection of fluid on portions of the boundary, and the objective function represents the rate at which energy is dissipated in the fluid. We develop reduced Hessian sequential quadratic programming methods that avoid converging the flow equations at each iteration. Both quasi-Newton and Newton variants are developed and compared to the approach of eliminating the flow equations and variables, which is effectively the generalized reduced gradient method. Optimal control problems are solved for two-dimensional flow around a cylinder and three-dimensional flow around a sphere. The examples demonstrate at least an order-of-magnitude reduction in time taken, allowing the optimal solution of flow control problems in as little as half an hour on a desktop workstation. © 1997 Academic Press

1. INTRODUCTION

Flow control has had a long history since Prandtl's early experiments demonstrated the feasibility of preventing flow separation by sucking fluid away from the boundary layer in a diverging channel [23]. Since then, much experimental work has been devoted to establishing the technological basis for flow control, and certain flow control problems have become amenable to analytical investigation, albeit often with simplifying assumptions; see the review by Gad-el-Hak [8].

Recently, interest has increased in *optimal flow control* of viscous fluids, that is, the determination of optimal values of controls based on the governing partial differential equations of the fluid, i.e., the Navier–Stokes equations [20]. These problems are among the most challenging optimization problems in computational science and engineering. They owe their complexity to their being constrained by numerical approximations of the Navier–Stokes equations. These constraints are highly nonlinear and can number in the millions. Conventional optimization approaches prove inadequate for such large-scale optimization problems.

The development of numerical optimization methods for optimal flow control is built on a mathematical foundation that continues to be enlarged. A number of basic results concerning existence and regularity of solutions to the continuous problem, as well as error estimates for its numerical approximation, have been established mostly over the last decade; see the article by Gunzburger *et al.* for a good overview [13].

This rich mathematical basis, the increasing power of computers, and the maturation of numerical methods for the flow simulation itself motivate the desire to develop numerical optimization methods for solution of optimal flow control problems. The latter forms the subject of this article. Here, we focus on a prototype problem of optimal control of fluids governed by the steady incompressible Navier–Stokes equations. The control is affected by the suction or injection of fluid on portions of the boundary, and the objective function represents the rate at which energy is dissipated in the fluid. We define the mathematical model in Section 2, and in Section 3 we develop reduced Hessian sequential quadratic programming (SQP) methods that exploit the structure of the optimality conditions and avoid converging the flow equations at each optimization iteration. Both quasi-Newton and Newton variants are developed. The SQP methods are compared in Section 4 to the approach of eliminating the flow equations and variables, which is effectively the generalized reduced gradient (GRG) method. The examples demonstrates at least an order-of-magnitude reduction in time taken, allowing the solution of some two-dimensional optimal flow control problems in around a half hour, and three-dimensional problems in reasonable time.

2. A PROBLEM IN OPTIMAL CONTROL OF NAVIER–STOKES FLOWS

Consider a steady, uniform, external, viscous, incompressible flow of a Newtonian fluid around a body with

[1] Supported in part by the Engineering Design Research Center, an Engineering Research Center of the National Science Foundation, under Grant EEC-8943164, and by Algor, Inc.

bounding surface Γ. We distinguish two possibly disjoint regions of the boundary: Γ_0, on which the velocity is specified to be zero (i.e., the no-slip condition is specified), and Γ_c, on which velocity controls are applied. Thus $\Gamma = \Gamma_0 \cup \Gamma_c$. To approximate the farfield velocity condition, we truncate the domain of the problem with an inflow boundary Γ_1 on which is enforced the freestream velocity $\mathbf{u}_\infty$, and an outflow boundary Γ_2 on which a zero-traction condition is maintained. The flow domain is denoted as Ω. Let us represent the velocity vector, pressure, stress tensor, density, and viscosity of the fluid by, respectively, $\mathbf{u}$, p, $\boldsymbol{\sigma}$, ρ, and μ.

The optimal control problem is to find the velocity control function $\mathbf{u}_c$ acting on Γ_c and the resulting fluid field variables that minimize the rate of energy dissipation, subject to the Navier–Stokes equations. Mathematically, the problem is to minimize

$$\frac{\mu}{2}\int_\Omega (\nabla\mathbf{u} + \nabla\mathbf{u}^T) : (\nabla\mathbf{u} + \nabla\mathbf{u}^T)\, d\Omega, \tag{2.1}$$

subject to

$$\rho(\mathbf{u}\cdot\nabla)\mathbf{u} - \nabla\cdot\boldsymbol{\sigma} = \mathbf{0} \quad \text{in } \Omega, \tag{2.2}$$

$$\boldsymbol{\sigma} = -\mathbf{I}p + \frac{\mu}{2}(\nabla\mathbf{u} + \nabla\mathbf{u}^T) \quad \text{in } \Omega, \tag{2.3}$$

$$\nabla\cdot\mathbf{u} = 0 \quad \text{in } \Omega, \tag{2.4}$$

$$\mathbf{u} = \mathbf{0} \quad \text{on } \Gamma_0, \tag{2.5}$$

$$\mathbf{u} = \mathbf{u}_c \quad \text{on } \Gamma_c, \tag{2.6}$$

$$\mathbf{u} = \mathbf{u}_\infty \quad \text{on } \Gamma_1, \tag{2.7}$$

$$\boldsymbol{\sigma}\cdot\mathbf{n} = \mathbf{0} \quad \text{on } \Gamma_2, \tag{2.8}$$

where $(\nabla\mathbf{u})_{ij} = \partial u_j/\partial x_i$, and the symbol ":" represents the scalar product of two tensors, so that

$$\nabla\mathbf{u} : \nabla\mathbf{v} = \sum_{i,j} \frac{\partial u_i}{\partial x_j}\frac{\partial v_i}{\partial x_j}.$$

Here, (2.1) is the dissipation function, (2.2) is the conservation of linear momentum equation, (2.3) is the constitutive law, (2.4) is the conservation of mass (actually volume) equation, and (2.5)–(2.8) are boundary conditions. We eliminate the constitutive law and stress by substituting (2.3) into (2.2), and we further reduce the size of the problem by employing a *penalty method:* let us relax (2.4) by replacing it with

$$\nabla\cdot\mathbf{u} = -\varepsilon p \quad \text{in } \Omega. \tag{2.9}$$

Clearly as $\varepsilon \to 0$, we recover the original equation; in fact, the error in the derivative of $\mathbf{u}$ is of order ε [11]. By introducing the pressure in the mass equation, we can eliminate it from the problem by solving for p in (2.9) and substituting the resulting expression into (2.3).

In general it is not possible to solve infinite dimensional optimization problems such as (2.1)–(2.8) in closed form. Thus, we seek numerical approximations. Here, we use a Galerkin finite element method. Let the Sobolev subspace $\mathscr{U}^h$ be the space of all C^0 continuous piecewise polynomials that vanish on Γ_1 and Γ_0, and define the Sobolev subspace $\mathscr{V}^h$ similarly, with the added requirement that the functions also vanish on Γ_c. By restricting the velocity and control vectors to $\mathscr{U}^h$, the infinite-dimensional optimization problem (2.1)–(2.8) becomes finite-dimensional. We "triangulate" the computational domain to obtain N_s nodes in Ω and on Γ_2, N_c nodes on Γ_c, and N_1 nodes on Γ_1. Corresponding to a node i with coordinates $\mathbf{x}_i$, we have the compactly supported finite element basis function $\phi_i(\mathbf{x})$. Let N represent the total number of unknown nodal velocity vectors in the optimal control problem, i.e., $N = N_s + N_c$. Thus,

$$\mathscr{U}^h = \operatorname{span}\{\phi_1, ..., \phi_N\},$$
$$\mathscr{V}^h = \operatorname{span}\{\phi_1, ..., \phi_{Ns}\},$$

Furthermore, we associate with each node in Ω, on Γ_2, and on Γ_c the nodal velocity vector $\mathbf{u}_i$. The unknown velocities, and hence optimization variables, consist of the nodal state velocities $\mathbf{u}_i$, $i = 1, ..., N_s$ and the nodal control velocities $\mathbf{u}_i$, $i = N_s + 1, ..., N$. Let $\mathbf{u}_\infty^h$ and $\mathbf{u}_c^h$ be the finite element interpolants of $\mathbf{u}_\infty$ and $\mathbf{u}_c$, i.e.,

$$\mathbf{u}_c^h(\mathbf{x}) = \sum_{i=N_s+1}^{N} \mathbf{u}_c(\mathbf{x}_i)\phi_i(\mathbf{x}),$$

and

$$\mathbf{u}_\infty^h(\mathbf{x}) = \sum_{i=N+1}^{N+N_1} \mathbf{u}_\infty(\mathbf{x}_i)\phi_i(\mathbf{x}).$$

Finally, we can represent $\mathbf{u}^h$, the finite element approximation to the velocity field, as

$$\mathbf{u}^h(\mathbf{x}) = \mathbf{u}_\infty^h(\mathbf{x}) + \sum_{i=1}^{N} \mathbf{u}_i\phi_i(\mathbf{x}),$$

where, because of the property of finite element basis functions, $\mathbf{u}^h(\mathbf{x}_i) = \mathbf{u}_i$. We note that the control function has been rendered finite dimensional; the control variables are simply the nodal values of the control function:

$$\mathbf{u}_i = \mathbf{u}_c(\mathbf{x}_i), \quad i = N_s + 1, ..., N.$$

The finite element approximation of the optimal control problem (2.1)–(2.8) can now be stated as finding $\mathbf{u}^h \in \mathcal{U}^h$ so that

$$\frac{\mu}{2}\int_\Omega (\nabla\mathbf{u}^h + (\nabla\mathbf{u}^h)^T) : (\nabla\mathbf{u}^h + (\nabla\mathbf{u}^h)^T)\, d\Omega \quad (2.10)$$

is minimized, subject to

$$\begin{aligned}&\frac{\mu}{2}\int_\Omega (\nabla\mathbf{u}^h + (\nabla\mathbf{u}^h)^T) : (\nabla\mathbf{v}^h + (\nabla\mathbf{v}^h)^T)\, d\Omega\\ &\quad+\frac{1}{\varepsilon}\int_\Omega (\nabla\cdot\mathbf{u}^h)(\nabla\cdot\mathbf{v}^h)\, d\Omega \qquad (2.11)\\ &\quad+\int_\Omega \mathbf{v}^h\cdot\rho(\mathbf{u}^h\cdot\nabla)\mathbf{u}^h\, d\Omega = 0 \quad \text{for all } \mathbf{v}^h \in \mathcal{V}^h.\end{aligned}$$

The objective function (2.10) is the discrete form of the dissipation function; the constraints (2.11) are a discretization of the variational form of the penalized conservation of linear momentum equation.

Let us define $n = dN$ as the total number of unknown velocity components, where d is the physical dimension of the flow, i.e., $d = 2$ or 3. Let $\mathbf{u} \in \Re^n$ denote the vector of unknown nodal velocity components.[2] The vector $\mathbf{u}$ includes both state and control variables and can be symbolically partitioned as

$$\mathbf{u} = \begin{Bmatrix}\mathbf{u}_s\\ \mathbf{u}_c\end{Bmatrix},$$

where $\mathbf{u}_s \in \Re^{n_s}$ represents the nodal velocities components associated with the state variables, and $\mathbf{u}_c \in \Re^{n_c}$ the nodal velocities corresponding to the controls. Similarly, we can partition $\mathbf{h}(\mathbf{u})$: $\Re^n \to \Re^n$, the discrete form of the nonlinear convective term in (2.2), into $\mathbf{h}_s(\mathbf{u})$: $\Re^n \to \Re^{n_s}$, a component associated with momentum equations written at nodes belonging to state variables, and $\mathbf{h}_c(\mathbf{u})$: $\Re^n \to \Re^{n_c}$, a control component, associated with control nodes:

$$\mathbf{h}(\mathbf{u}) = \begin{Bmatrix}\mathbf{h}_s(\mathbf{u})\\ \mathbf{h}_c(\mathbf{u})\end{Bmatrix}.$$

As can be seen from (2.11), $\mathbf{h}(\mathbf{u})$ is quadratic in $\mathbf{u}$. Finally, we define the matrix $\mathbf{K}^\mu \in \Re^{n\times n}$ arising from the discrete form of the viscous term, and $\mathbf{K}^\varepsilon \in \Re^{n\times n}$ corresponding to the discrete "pressure" term, in the momentum equation. Both $\mathbf{K}^\mu$ and $\mathbf{K}^\varepsilon$ are symmetric positive definite. These matrices can be partitioned into blocks corresponding to state and control variables,

$$\mathbf{K}^\mu = \begin{bmatrix}\mathbf{K}^\mu_{ss} & \mathbf{K}^\mu_{sc}\\ \mathbf{K}^\mu_{cs} & \mathbf{K}^\mu_{cc}\end{bmatrix}, \quad \mathbf{K}^\varepsilon = \begin{bmatrix}\mathbf{K}^\varepsilon_{ss} & \mathbf{K}^\varepsilon_{sc}\\ \mathbf{K}^\varepsilon_{cs} & \mathbf{K}^\varepsilon_{cc}\end{bmatrix}.$$

The sparsity of $\mathbf{K}^\mu$, $\mathbf{K}^\varepsilon$, and the Jacobian of $\mathbf{h}(\mathbf{u})$ is dictated by the sparsity of the graph underlying the finite element mesh. For quasi-uniform meshes, a node has a bounded number of neighbors, so that the number of nonzeroes per row is independent of problem size. Thus, these matrices have $\mathcal{O}(n)$ nonzeroes. The constant is determined by d, by the order of the basis functions $\phi(\mathbf{x})$, and by the structure of the mesh, but is usually small.

The optimization problem (2.10)–(2.11) can be rewritten as follows: find $\mathbf{u}_s \in \Re^{n_s}$ and $\mathbf{u}_c \in \Re^{n_c}$ so that

$$\tfrac{1}{2}\mathbf{u}_s^T\mathbf{K}^\mu_{ss}\mathbf{u}_s + \tfrac{1}{2}\mathbf{u}_c^T\mathbf{K}^\mu_{cc}\mathbf{u}_c + \mathbf{u}_s^T\mathbf{K}^\mu_{sc}\mathbf{u}_c \quad (2.12)$$

is minimized, subject to

$$(\mathbf{K}^\mu_{ss} + \mathbf{K}^\varepsilon_{ss})\,\mathbf{u}_s + (\mathbf{K}^\mu_{sc} + \mathbf{K}^\varepsilon_{sc})\,\mathbf{u}_c + \mathbf{h}_s(\mathbf{u}) = \mathbf{0}. \quad (2.13)$$

The problem (2.12)–(2.13) is characterized by $n = n_s + n_c$ variables and n_s nonlinear equality constraints. Both the objective function and constraints are quadratic in the optimization variables. For practical problems, $n_s \gg n_c$, since the control is affected at a small number of boundary points, while the state variables are associated with a triangulation of the flow domain. The number of *degrees of freedom* of the optimization problem, i.e., the number of variables n less the number of independent constraints n_s, is equal to the number of control variables and is thus small relative to the size of the problem. For problems of industrial interest, values as high as $n_s = \mathcal{O}(10^6)$, $n_c = \mathcal{O}(10^3)$ are desirable, although such problems are currently beyond the range of existing computers. We are thus in the realm of extremely large-scale, sparsely constrained nonlinear optimization. In the next section we develop SQP methods for this problem.

3. SQP METHODS

The most popular approach to solving optimization problems that are constrained by discretized partial differential equations (PDEs) is to eliminate the constraints by solving them for the state variables, given values of the control variables. For example, solve for $\mathbf{u}_s$ in (2.13) as a function of $\mathbf{u}_c$, and substitute this into the objective (2.12).[3] The state variables are thus eliminated from the problem,

[2] Not to be confused with $\mathbf{u}(\mathbf{x}) \in \Re^3$, the exact velocity field; the distinction will be clear from the context.

[3] Of course, this substitution is only implicit: ones solves (2.13) numerically at each optimization iteration, given current values of $\mathbf{u}_c$.

and we now have an unconstrained optimization problem in the n_c variables $\mathbf{u}_c$. The gradient of the objective can be obtained through the implicit function theorem. This method is essentially the GRG method, since we are satisfying the constraints at each iteration [10].

Advantages of GRG for PDE-constrained optimization problems include: (i) a large reduction in size of the problem, especially when $n_c \ll n_s$, as is common in optimal control or optimal design; (ii) avoiding the large, sparse Hessians matrices inherent in the formulation as a constrained problem of the form (2.12)–(2.13), in favor of a small, dense Hessian, thus enabling the use of standard dense nonlinear optimization software; (iii) the ability to use existing PDE solution algorithms[4] in eliminating the state equations given values of control variables (i.e., the *forward problem*). This last advantage should not be taken lightly—over the past decade, many specialized and sophisticated algorithms (e.g., multilevel methods and preconditioned Krylov subspace methods) have been developed for solving various classes of PDEs and incorporated into robust software. If we retain the discrete PDEs as constraints, the optimizer becomes responsible for converging the state equations. Thus it is not immediately obvious how to extend sophisticated PDE solution techniques, such as domain decomposition methods and multilevel preconditioners, to the optimization problem, as solution of (2.12)–(2.13) appears to require. For these reasons, state equation elimination methods are almost universal for PDE-constrained problems. For examples of this approach in the context of flow control, see the papers contained in [12].

Nevertheless, the approach outlined in the last two paragraphs possesses a distinct disadvantage: it requires exact solution of the state equations at *each* iteration. This can be an onerous requirement, especially for highly nonlinear problems such as those governed by Navier–Stokes equations. Instead, we pursue here *bona fide* SQP methods for the problem (2.12)–(2.13). SQP requires satisfaction of only a *linear* approximation of the constraints at each iteration, thereby avoiding the need to converge them fully [10]. Thus, state equations are satisfied simultaneously as control variables are converged to their optimal values. Furthermore, we consider a special reduced Hessian SQP method that retains the three advantages attributed to GRG in the paragraph above. In order to retain the last advantage, i.e., that existing (GRG-ready) PDE solvers can be used, several conditions must be met. First, the PDE solver must be Newton-based. Second, the sensitivity analysis capability of the PDE solver must be based on an adjoint approach. Finally, the solver must be modular enough so that the inner Newton linear step can be isolated from the code.

[4] With the addition of provisions for computing the gradient of the objective function (i.e., sensitivity analysis).

We begin with some definitions. Let $\mathbf{c}_s(\mathbf{u})$: $\Re^n \to \Re^{n_s}$ represent the residual of the discrete Navier–Stokes equations,

$$\mathbf{c}_s(\mathbf{u}) \equiv (\mathbf{K}^{\mu}_{ss} + \mathbf{K}^{\varepsilon}_{ss})\mathbf{u}_s + (\mathbf{K}^{\mu}_{sc} + \mathbf{K}^{\varepsilon}_{sc})\mathbf{u}_c + \mathbf{h}_s(\mathbf{u}),$$

and let $\mathbf{J}(\mathbf{u}) \in \Re^{n\times n}$ denote the Jacobian of the nonlinear convective term $\mathbf{h}(\mathbf{u})$ with respect to the velocities $\mathbf{u}$. The matrix $\mathbf{J}(\mathbf{u})$ is nonsymmetric and indefinite and can be partitioned into state and control blocks, in the same manner as the viscous and pressure matrices. Note that the sparsity structure of $\mathbf{J}(\mathbf{u})$ is identical to that of $\mathbf{K}^{\mu}$. Let the matrix $\mathbf{A}_s(\mathbf{u}) \in \Re^{n_s\times n}$ represent the Jacobian of $\mathbf{c}_s(\mathbf{u})$ with respect to $\mathbf{u}$. We can partition $\mathbf{A}_s(\mathbf{u})$ into $\mathbf{A}_{ss}(\mathbf{u}) \in \Re^{n_s\times n_s}$, the Jacobian of $\mathbf{c}_s(\mathbf{u})$ with respect to $\mathbf{u}_s$, and $\mathbf{A}_{sc}(\mathbf{u}) \in \Re^{n_s\times n_c}$, the Jacobian of $\mathbf{c}_s(\mathbf{u})$ with respect to $\mathbf{u}_c$. Explicitly,

$$\mathbf{A}_s(\mathbf{u}) = [\mathbf{A}_{ss}(\mathbf{u}), \mathbf{A}_{sc}(\mathbf{u})],$$

with

$$\mathbf{A}_{ss}(\mathbf{u}) \equiv \mathbf{K}^{\mu}_{ss} + \mathbf{K}^{\varepsilon}_{ss} + \mathbf{J}_{ss}(\mathbf{u}),$$
$$\mathbf{A}_{sc}(\mathbf{u}) \equiv \mathbf{K}^{\mu}_{sc} + \mathbf{K}^{\varepsilon}_{sc} + \mathbf{J}_{sc}(\mathbf{u}).$$

Thus $\mathbf{A}_{ss}(\mathbf{u})$ and $\mathbf{A}_{sc}(\mathbf{u})$ have the same sparsity structure as $\mathbf{K}^{\mu}_{ss}$ and $\mathbf{K}^{\mu}_{sc}$, respectively.

We define the Lagrangian function, $\mathscr{L}(\mathbf{u}, \boldsymbol{\lambda}_s)$, for the optimization problem (2.12)–(2.13), as

$$\begin{aligned}\mathscr{L}(\mathbf{u}, \boldsymbol{\lambda}_s) \equiv {} & \tfrac{1}{2}\mathbf{u}_s^T\mathbf{K}^{\mu}_{ss}\mathbf{u}_s + \tfrac{1}{2}\mathbf{u}_c^T\mathbf{K}^{\mu}_{cc}\mathbf{u}_c + \mathbf{u}_s^T\mathbf{K}^{\mu}_{sc}\mathbf{u}_c \\ & + \boldsymbol{\lambda}_s^T(\mathbf{K}^{\mu}_{ss} + \mathbf{K}^{\varepsilon}_{ss})\mathbf{u}_s + \boldsymbol{\lambda}_s^T(\mathbf{K}^{\mu}_{sc} + \mathbf{K}^{\varepsilon}_{sc})\mathbf{u}_c + \boldsymbol{\lambda}_s^T\mathbf{h}_s(\mathbf{u}),\end{aligned} \tag{3.1}$$

where $\boldsymbol{\lambda}_s \in \Re^n_s$ is the vector of Lagrange multipliers corresponding to the state equations. Let $\mathbf{g}(\mathbf{u}) \equiv \mathbf{K}^{\mu}\mathbf{u}$ represent the gradient of the objective function (2.12). We denote by $\mathbf{H}_k \in \Re^{n\times n}$ the symmetric, indefinite Hessian matrix of the kth element of $\mathbf{h}(\mathbf{u})$, and by $\mathbf{W}(\lambda_s) \in \Re^{n\times n}$ the symmetric, indefinite Hessian matrix of the Lagrangian function $\mathscr{L}(\mathbf{u}, \boldsymbol{\lambda})$ (both Hessians are with respect to $\mathbf{u}$). Because $\mathbf{h}(\mathbf{u})$ is quadratic in $\mathbf{u}$, $\mathbf{H}_k$ is a constant matrix. Note that $(\mathbf{H}_k)_{ij}$ is nonzero only when nodes i, j, and k all belong to the same element; it follows that the (i, j) entry of the matrix

$$\mathbf{H}(\boldsymbol{\lambda}) \equiv \sum_{k=1}^{n_s} \lambda_k \mathbf{H}_k$$

is nonzero only when nodes i and j belong to the same element. Therefore, $\mathbf{H}(\boldsymbol{\lambda})$ has the same mesh-based sparsity structure as the other finite element matrices ($\mathbf{K}^{\mu}$, $\mathbf{K}^{\varepsilon}$, and $\mathbf{J}$), as does $\mathbf{W}(\boldsymbol{\lambda})$, since $\mathbf{W}(\boldsymbol{\lambda}) = \mathbf{K}^{\mu} + \mathbf{H}(\boldsymbol{\lambda})$,

The necessary conditions for an optimal solution of the problem (2.12)–(2.13) reflect the vanishing of the gradient of the Lagrangian (3.1) and can be expressed by the set of nonlinear equations

$$\begin{bmatrix} \mathbf{K}^{\mu}_{ss} & \mathbf{K}^{\mu}_{sc} & \mathbf{K}^{\mu}_{ss} + \mathbf{K}^{\varepsilon}_{ss} \\ \mathbf{K}^{\mu}_{cs} & \mathbf{K}^{\mu}_{cc} & \mathbf{K}^{\mu}_{cs} + \mathbf{K}^{\varepsilon}_{cs} \\ \mathbf{K}^{\mu}_{ss} + \mathbf{K}^{\varepsilon}_{ss} & \mathbf{K}^{\mu}_{sc} + \mathbf{K}^{\varepsilon}_{sc} & \mathbf{0} \end{bmatrix} \begin{Bmatrix} \mathbf{u}_s \\ \mathbf{u}_c \\ \boldsymbol{\lambda}_s \end{Bmatrix} + \begin{Bmatrix} \mathbf{J}^T_{ss}\boldsymbol{\lambda}_s \\ \mathbf{J}^T_{sc}\boldsymbol{\lambda}_s \\ \mathbf{h}_s(\mathbf{u}) \end{Bmatrix} = \begin{Bmatrix} \mathbf{0} \\ \mathbf{0} \\ \mathbf{0} \end{Bmatrix}. \tag{3.2}$$

SQP can be derived as a Newton method for solving the optimality conditions [10]. One step of Newton's method for (3.2) is given by solving the linear system

$$\begin{bmatrix} \mathbf{K}^{\mu}_{ss} + \mathbf{H}^{k}_{ss} & \mathbf{K}^{\mu}_{sc} + \mathbf{H}^{k}_{sc} & \mathbf{K}^{\mu}_{ss} + \mathbf{K}^{\varepsilon}_{ss} + (\mathbf{J}^{k}_{ss})^T \\ \mathbf{K}^{\mu}_{cs} + \mathbf{H}^{k}_{cs} & \mathbf{K}^{\mu}_{cc} + \mathbf{H}^{k}_{cc} & \mathbf{K}^{\mu}_{sc} + \mathbf{K}^{\varepsilon}_{sc} + (\mathbf{J}^{k}_{sc})^T \\ \mathbf{K}^{\mu}_{ss} + \mathbf{K}^{\varepsilon}_{ss} + \mathbf{J}^{k}_{ss} & \mathbf{K}^{\mu}_{sc} + \mathbf{K}^{\varepsilon}_{sc} + \mathbf{J}^{k}_{sc} & \mathbf{0} \end{bmatrix} \begin{Bmatrix} \mathbf{p}^k_s \\ \mathbf{p}^k_c \\ \boldsymbol{\lambda}^{k+1}_s \end{Bmatrix} = - \begin{Bmatrix} \mathbf{K}^{\mu}_{ss}\mathbf{u}^k_s + \mathbf{K}^{\mu}_{sc}\mathbf{u}^k_c \\ \mathbf{K}^{\mu}_{cs}\mathbf{u}^k_s + \mathbf{K}^{\mu}_{cc}\mathbf{u}^k_c \\ (\mathbf{K}^{\mu}_{ss} + \mathbf{K}^{\varepsilon}_{ss})\mathbf{u}^k_s + (\mathbf{K}^{\mu}_{sc} + \mathbf{K}^{\varepsilon}_{sc})\mathbf{u}^k_c + \mathbf{h}^k_s = \mathbf{0}, \end{Bmatrix} \tag{3.3}$$

for the increments in the state and control velocities, $\mathbf{p}^k_s$ and $\mathbf{p}^k_c$, and for the new multiplier estimate $\boldsymbol{\lambda}^{k+1}_s$, where the superscript k indicates evaluation of a quantity at $\mathbf{u}^k$ or $\boldsymbol{\lambda}^k$. The state and control variables are then updated by

$$\mathbf{u}^{k+1}_s = \mathbf{u}^k_s + \mathbf{p}^k_s, \quad \mathbf{u}^{k+1}_c = \mathbf{u}^k_c + \mathbf{p}^k_c.$$

Let us rewrite the Newton equations (3.3) in terms of the symbols defined at the beginning of this section as

$$\begin{bmatrix} \mathbf{W}^k_{ss} & \mathbf{W}^k_{sc} & (\mathbf{A}^k_{ss})^T \\ \mathbf{W}^k_{cs} & \mathbf{W}^k_{cc} & (\mathbf{A}^k_{sc})^T \\ \mathbf{A}^k_{ss} & \mathbf{A}^k_{sc} & \mathbf{0} \end{bmatrix} \begin{Bmatrix} \mathbf{p}^k_s \\ \mathbf{p}^k_c \\ \boldsymbol{\lambda}^{k+1}_s \end{Bmatrix} = \begin{Bmatrix} -\mathbf{g}^k_s \\ -\mathbf{g}^k_c \\ -\mathbf{c}^k_s \end{Bmatrix}. \tag{3.4}$$

The major difficulty in solving this linear system is its extremely large, sparse coefficient matrix, the Karush–Kuhn–Tucker (KKT) matrix, which is of order $(n + n_s) \times (n + n_s)$. For industrial flow problems on unstructured meshes, the forward (i.e., flow simulation) problem is memory-bound, and even on large supercomputers one can barely afford memory for $\mathbf{A}_{ss}$, let alone the entire matrix. So sparse factorization of the KKT matrix is not viable. One possibility is to solve (3.4) using a Krylov method such as the minimum residual method. However, it is not immediately obvious how to precondition the coefficient matrix.

Instead, consider the following block elimination. In the remainder of this section, we drop the superscript k; it is understood that all quantities depending on $\mathbf{u}$ or $\boldsymbol{\lambda}_s$ are evaluated at $\mathbf{u}^k$ or $\boldsymbol{\lambda}^k_s$. First, solve for $\mathbf{p}_s$ from the last block of equations to obtain

$$\mathbf{p}_s = -\mathbf{A}^{-1}_{ss}(\mathbf{c}_s + \mathbf{A}_{sc}\mathbf{p}_c), \tag{3.5}$$

where the invertibility of $\mathbf{A}_{ss}$ is guaranteed by the well-posedness of the boundary value problem (provided of course that we are away from limit points). Then, substitute this expression into the first block of equations. Finally, premultiply the first block of equations by $-\mathbf{A}^T_{sc}\mathbf{A}^{-T}_{ss}$ and add it to the second block of equations, thereby eliminating $\boldsymbol{\lambda}_s$. The result is a linear system that can be solved for $\mathbf{p}_c$, namely

$$\begin{aligned} \mathbf{W}_z\mathbf{p}_c &= (\mathbf{A}^T_{sc}\mathbf{A}^{-T}_{ss}\mathbf{W}_{ss} - \mathbf{W}_{cs})\mathbf{A}^{-1}_{ss}\mathbf{c}_s + \mathbf{A}^T_{sc}\mathbf{A}^{-T}_{ss}\mathbf{g}_s - \mathbf{g}_c, \\ \mathbf{W}_z &\equiv (\mathbf{A}^T_{sc}\mathbf{A}^{-T}_{ss}\mathbf{W}_{ss}\mathbf{A}^{-1}_{ss}\mathbf{A}_{sc} - \mathbf{A}^T_{sc}\mathbf{A}^{-T}_{ss}\mathbf{W}_{sc} \\ &\quad - \mathbf{W}_{cs}\mathbf{A}^{-1}_{ss}\mathbf{A}_{sc} + \mathbf{W}_{cc}). \end{aligned} \tag{3.6}$$

Finally, the new estimate of the Lagrange multiplier vector is recovered from

$$\boldsymbol{\lambda}^{k+1}_s = -\mathbf{A}^{-T}_{ss}(\mathbf{W}_{ss}\mathbf{p}_s + \mathbf{W}_{sc}\mathbf{p}_c + \mathbf{g}_s), \tag{3.7}$$

which is a second-order multiplier estimate. This leads to the following algorithm.

Algorithm 3.1 [Newton SQP].

```
$k = 0; \mathbf{u}^0_s = \mathbf{u}^0_c = \boldsymbol{\lambda}^0_s = \mathbf{0}$
while $\| (\mathbf{A}^k_s)^T \boldsymbol{\lambda}^k_s - \mathbf{g}^k \| \neq 0$ and $\| \mathbf{c}^k_s \| \neq 0$
    $k = k + 1$
    Solve (3.6) for $\mathbf{p}^k_c$
    $\mathbf{u}^{k+1}_c = \mathbf{u}^k_c + \mathbf{p}^k_c$
    Find $\mathbf{p}^k_s$ from (3.5)
    $\mathbf{u}^{k+1}_s = \mathbf{u}^k_s + \mathbf{p}^k_s$
    Find $\boldsymbol{\lambda}^{k+1}_s$ from (3.7)
end
```

Algorithm 3.1 displays a quadratic convergence rate provided that (i) at the optimal solution we are away from a limit or bifurcation point in the forward problem (i.e., $\mathbf{A}_{ss}$ is nonsingular); (ii) $\mathbf{W}_z$ is positive definite at the optimal solution; and (iii) ($\mathbf{u}^0$, $\boldsymbol{\lambda}_s^0$) is sufficiently close to the optimal solution.

The justification for the block elimination (3.5)–(3.7) and the resulting Algorithm 3.1 is that there result only two types of linear system to be solved: those involving $\mathbf{A}_{ss}$ or its transpose as the coefficient matrix, and the system that determines $\mathbf{p}_c$, (3.6), with coefficient matrix $\mathbf{W}_z$. In the former case, these systems are "easy" to solve, since they have the same coefficient matrix as that of a Newton step for the state equations. Thus any (Newton-based) Navier–Stokes solver can be enlisted for this task, enabling the exploitation of many of the advances in solving the forward problem developed over the last decade (including domain decomposition and multilevel methods). In the latter case, solution of (3.6) is also easy since $\mathbf{W}_z$ is of order of the number of control variables, which under the assumption that $n_s \gg n_c$, is very small. Standard dense factorization is therefore appropriate.

When implementing Algorithm 3.1, one of course does not invert $\mathbf{A}_{ss}$; one instead forms the matrix $\mathbf{A}_{ss}^{-1}\mathbf{A}_{sc}$ by solving, with coefficient matrix $\mathbf{A}_{ss}$, for the n_c right-hand sides composed of the columns of $\mathbf{A}_{sc}$. An additional solve with the same coefficient matrix for the right-hand side $\mathbf{c}_s$ is necessary. Finally, (3.7) implies an additional right-hand side solve, but with the transpose of $\mathbf{A}_{ss}$ as coefficient matrix. So each iteration of Algorithm 3.1 requires solving a linear system with coefficient matrix $\mathbf{A}_{ss}$ (the state equation Jacobian matrix) and having $n_c + 1$ right-hand sides, as well as a linear system with $\mathbf{A}_{ss}^T$ as coefficient matrix and one right-hand side. If sparse factorization of $\mathbf{A}_{ss}$ is viable, for example for two-dimensional flows or low Reynolds number three-dimensional flows, then one iteration of Algorithm 3.1 entails one factorization and $n_c + 2$ pairs of triangular solves (compare this with full solution of the flow equations, as in GRG). If quasi-uniform meshes and nested dissection orderings are used, and if pivoting is not required, $\mathbf{A}_{ss}$ can be factored with $\mathcal{O}(n_s^2)$ work in 3D and $\mathcal{O}(n_s^{1.5})$ work in 2D [18]. In any case, the cost of one iteration of Algorithm 3.1 is a fraction of the cost of the forward problem. On the other hand, if sparse factorization is not practical, and an iterative method must be used, one is faced with $n_c + 2$ solves. When n_c is large, it becomes imperative to use iterative methods tailored to multiple right-hand sides; it also pays to invest in a good preconditioner, since its construction can be amortized over the right-hand sides. In particular, domain decomposition methods tailored to multiple right-hand sides appear to be attractive [6].

Once the matrix $\mathbf{A}_{ss}^{-1}\mathbf{A}_{sc}$ is created, forming its products with submatrices of $\mathbf{W}$ presents no difficulty. Recall that $\mathbf{W}$ has $\mathcal{O}(n)$ nonzeroes and sparsity structure dictated by the underlying finite element mesh. In particular it is stored using the same (sparse compressed row) data structure that all the other finite element matrices use. Therefore forming products with submatrices of $\mathbf{W}$ requires work proportional to the row dimension of the submatrix. Thus, it is easy to see that, beyond forming the matrix $\mathbf{A}_{ss}^{-1}\mathbf{A}_{sc}$, the only other major effort in Algorithm 3.1 is $\mathcal{O}(n_s n_c^2)$ work associated with forming $\mathbf{W}_z$, and $\mathcal{O}(n_c^3)$ in factoring $\mathbf{W}_z$.

Let us examine the connection with other SQP methods. In fact, the block elimination (3.5)–(3.7) is identical to a reduced Hessian SQP method with a particular choice of null and range space bases. This can be seen by decomposing the search direction $\mathbf{p}$ into two components,

$$\mathbf{p} = \mathbf{Z}\mathbf{p}_z + \mathbf{Y}\mathbf{p}_y, \tag{3.8}$$

in which $\mathbf{Z} \in \Re^{n \times n_c}$ is a matrix whose columns form a basis for the null space of $\mathbf{A}_s$, and $\mathbf{Y} \in \Re^{n \times n_s}$ is chosen so that the matrix

$$\mathbf{Q} = [\mathbf{Z}\ \mathbf{Y}]$$

is nonsingular, and hence $\mathbf{Z}$ and $\mathbf{Y}$ form a basis for $\Re^n$. We refer to $\mathbf{p}_y$ as the *range space* component, even though strictly speaking, the columns of $\mathbf{Y}$ need not span the range space of $\mathbf{A}_s^T$.

The range space step is completely determined by substituting (3.8) into the last block of (3.4), resulting in the $n_s \times n_s$ system

$$\mathbf{A}_s\mathbf{Y}\mathbf{p}_y = -\mathbf{c}_s. \tag{3.9}$$

The null space move is found by substituting (3.8) into the first two blocks of (3.4), and premultiplying by $\mathbf{Z}^T$, to obtain the equations for $\mathbf{p}_z$,

$$\mathbf{Z}^T\mathbf{W}\mathbf{Z}\mathbf{p}_z = -\mathbf{Z}^T(\mathbf{g} + \mathbf{W}\mathbf{Y}\mathbf{p}_y). \tag{3.10}$$

The $n_c \times n_c$ matrix $\mathbf{Z}^T\mathbf{W}\mathbf{Z}$ is known as the *reduced Hessian* matrix. If one chooses the nonorthogonal bases

$$\mathbf{Z} = \begin{bmatrix} -\mathbf{A}_{ss}^{-1}\mathbf{A}_{sc} \\ \mathbf{I} \end{bmatrix}, \tag{3.11}$$

and

$$\mathbf{Y} = \begin{bmatrix} \mathbf{I} \\ \mathbf{0} \end{bmatrix} \tag{3.12}$$

then one sees that the null space step (3.10) is identical to (3.6), the equation for determining the move in the control variables. Indeed, the coefficient matrix of (3.6), $\mathbf{W}_z$, is exactly the reduced Hessian $\mathbf{Z}^T\mathbf{W}\mathbf{Z}$ and is therefore at least positive semidefinite in the vicinity of a minimum. The state variable update (3.5) is comprised of the state equation Newton step, i.e., (3.9) using the range basis

(3.12), as well as the null space contribution $-\mathbf{A}_{ss}^{-1}\mathbf{A}_{sc}\mathbf{p}_z$, where $\mathbf{p}_z \equiv \mathbf{p}_c$. The choice of bases (3.11) and (3.12) is known as a "coordinate basis" and has been applied to optimization problems in inverse heat conduction [19], structural design [24, 25], and compressible flow [21, 22]. SQP methods using these bases have been analyzed in [2, 7, 26], among others.

As mentioned earlier, one of the difficulties with Algorithm 3.1, i.e., the *bona fide* Newton method, arises when iterative solution of systems involving $\mathbf{A}_{ss}$ is necessary; in this case the benefit from solving $n_c + 2$ systems with the same coefficient matrix but different right-hand side is not as extensive as with sparse LU factorization. Might it be possible to give up the (local) quadratic convergence guarantee in exchange for the need to solve fewer systems involving $\mathbf{A}_{ss}$ at each iteration?

The answer turns out to be affirmative if we consider a quasi-Newton, rather than a true Newton, method. Consider (3.6), the control variable equation. Its coefficient matrix is the reduced Hessian $\mathbf{W}_z$. This matrix is positive definite at a strict local minimum, as well as being small ($n_c \times n_c$) and dense. It makes sense to recur a quasi-Newton approximation to it; thus we can avoid the construction of the matrix $\mathbf{A}_{ss}^{-1}\mathbf{A}_{sc}$. By replacing $\mathbf{W}_z$ with its quasi-Newton approximation, $\mathbf{B}_z$, it is easy to see that (3.5)–(3.7) now entail only five solutions of systems with $\mathbf{A}_{ss}$ or its transpose as coefficient matrix. This is a big reduction, especially for large n_c. However, we can do even better. At the expense of a reduction from one-step to two-step superlinear convergence [2], we ignore the second-order terms (those involving submatrices of $\mathbf{W}$) on the right-hand side of (3.6). Furthermore, we reduce (3.7) to a first-order Lagrange multiplier estimate by dropping terms involving blocks of $\mathbf{W}$. This results in the following algorithm, in which the BFGS formula is used to update the quasi-Newton approximation of the reduced Hessian.

ALGORITHM 3.2 [Quasi-Newton SQP].

```
$k = 0; \mathbf{u}_s^0 = \mathbf{u}_c^0 = \boldsymbol{\lambda}_s^0 = \mathbf{0}; \mathbf{B}_z^0 = \mathbf{I}$
$\boldsymbol{\lambda}_s^0 = (\mathbf{A}_{ss}^0)^{-T}\mathbf{g}_s^0$
$\mathbf{g}_z^0 = -(\mathbf{A}_{sc}^0)^T\boldsymbol{\lambda}_s^0 + \mathbf{g}_c^0$
**while** $\|\mathbf{g}_z^k\| \neq 0$ and $\|\mathbf{c}_s^k\| \neq 0$
    $\mathbf{p}_c^k = -(\mathbf{B}_z^k)^{-1}\mathbf{g}_z$
    $\mathbf{u}_c^{k+1} = \mathbf{u}_c^k + \mathbf{p}_c^k$
    $\mathbf{p}_s^k = -(\mathbf{A}_{ss}^k)^{-1}(\mathbf{c}_s^k + \mathbf{A}_{sc}^k\mathbf{p}_c^k)$
    $\mathbf{u}_s^{k+1} = \mathbf{u}_s^k + \mathbf{p}_s^k$
    $\boldsymbol{\lambda}_s^{k+1} = (\mathbf{A}_{ss}^{k+1})^{-T}\mathbf{g}_s^{k+1}$
    $\mathbf{g}_z^{k+1} = -(\mathbf{A}_{sc}^{k+1})^T\lambda_s^{k+1} + \mathbf{g}_c^{k+1}$
    $\mathbf{y}_z^k = \mathbf{g}_z^{k+1} - \mathbf{g}_z^k$
    $\mathbf{B}_z^{k+1} = \mathbf{B}_z^k + \frac{1}{(\mathbf{g}_z^k)^T\mathbf{p}_c^k}\mathbf{g}_z^k(\mathbf{g}_z^k)^T + \frac{1}{(\mathbf{y}_z^k)^T\mathbf{p}_c^k}\mathbf{y}_z^k(\mathbf{y}_z^k)^T$
    $\mathbf{g}_z^k = \mathbf{g}_z^{k+1}$
    $k = k + 1$
**end**
```

Algorithm 3.2 requires only *two* solves involving $\mathbf{A}_{ss}$ per iteration, as compared with $n_c + 2$ in Algorithm 3.1. This represents a substantial reduction in effort when iterative solvers are used and when n_c is significant. Of course, Algorithm 3.2 will generally require more iterations to converge than Algorithm 3.1, since it does not compute exact curvature information. The first of the two linear solves has $\mathbf{A}_{ss}$ as its coefficient matrix, and we term it the *state variable update.* It comprises two components: a Newton step on the state equations ($-\mathbf{A}_{ss}^{-1}\mathbf{c}_s$), and a first-order change in the states due to a change in the control variables ($-\mathbf{A}_{ss}^{-1}\mathbf{A}_{sc}\mathbf{p}_c$). The second linear system to be solved at each iteration has $\mathbf{A}_{ss}^T$ as its coefficient matrix, and is termed the *adjoint* step, because of parallels with adjoint methods for sensitivity analysis [14, 15]. The steps of this algorithm are almost identical to a quasi-Newton GRG method; the major difference is that in GRG the state equations are fully converged at each iteration, while in Algorithm 3.2 essentially only a single Newton step is carried out.

Algorithms 3.1 and 3.2 as presented above are not sufficient to guarantee convergence to a stationary point from arbitrary initial points. It is well known that for the forward problem, i.e., solving the discrete Navier–Stokes equations, Newton's method is only locally convergent. The diameter of the ball of convergence is of the order of the inverse of the Reynolds number that characterizes the flow [11]; better initial guesses are thus required as the Reynolds number increases. The optimal control problem should be no easier to converge than the forward problem, given that the flow equations form part of the first-order optimality conditions. This suggests continuation methods, which are popular techniques for globalizing the forward problem [11]. Here, we use a simple continuation on Reynolds number. That is, suppose we want to solve an optimal control problem with a Reynolds number of Re*, for which it is difficult to find a starting point from which Newton's method will converge. Instead, we solve a sequence of optimization problems characterized by increasing Reynolds number, beginning with Re = 0, and incrementing by ΔRe. Optimization problem i, with Reynolds number $i\Delta\text{Re}/\text{Re}^*$, is solved by either Algorithm 3.1 or 3.2, to generate a good starting point for problem $i + 1$. Algorithm 3.1 is initialized with the optimal Lagrange multipliers and state and control variables from optimization problem $i - 1$. Algorithm 3.2 includes the same initializations, but in addition takes the initial BFGS approximation to the Lagrangian Hessian matrix to be the Hessian approximation at the solution of problem $i - 1$. Note that when Re = 0, the nonlinear terms drop from the flow equations, and thus optimization problem (2.12)–(2.13) is an equality-constrained quadratic programming problem, solvable in one step. For subsequent problems, there exists a sufficiently small ΔRe such that Algorithms 3.1 and 3.2 con-

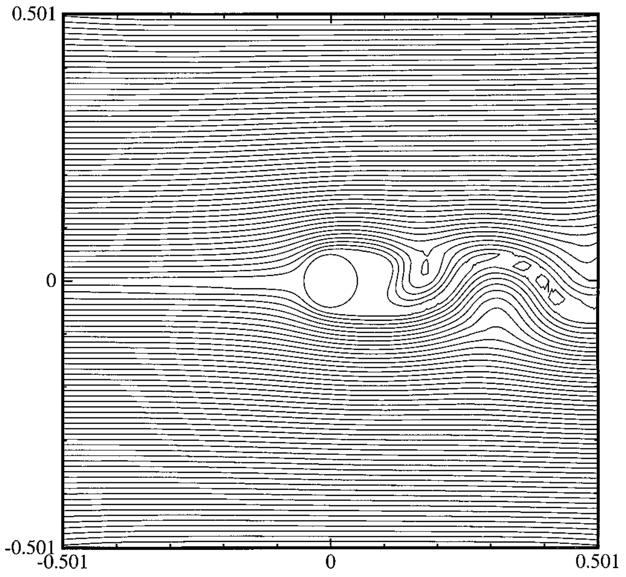

FIG. 4.1. Time-dependent streamlines, no control, Re = 500.

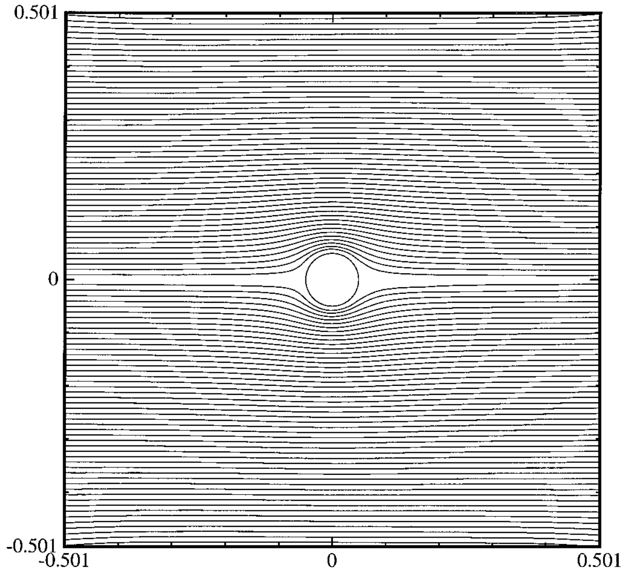

FIG. 4.2. Time-dependent streamlines, optimal control, Re = 500, t = 3.5 s.

verge to the solution of optimization problem $i + 1$ using initial data from problem i (provided we are away from bifurcation or singular points).

In the next section we use continuation variants of Algorithms 3.1 and 3.2 to solve some model problems in boundary control of viscous incompressible flow, and compare their performance to the GRG methods.

4. NUMERICAL EXAMPLES

In this section we compare Algorithm 3.1 (Newton-SQP, or N-SQP) and Algorithm 3.2 (quasi-Newton-SQP, or QN-SQP) of the previous section with both a quasi-Newton-GRG method (QN-GRG) and a steepest descent-GRG method (SD-GRG). With the QN-GRG method we converge the flow equations fully at each optimization iteration using a Newton solver, and we employ a BFGS formula to approximate the Hessian of the objective function.[5] SD-GRG refers to a similar method, except that a search direction is taken in a direction opposite to the gradient of the objective function. This method is chosen because of its popularity in the optimal control literature and since it is easy to implement. In both GRG cases, the sensitivity equations are used to compute the objective function gradient exactly using a direct (as opposed to adjoint) method (see, e.g., [14] or [15]).

Continuation is applied to N-SQP and QN-SQP as described at the end of Section 3. We apply continuation to both GRG methods at the level of the forward problem: since the GRG methods entail satisfaction of the flow equations at each iteration, the forward problem is completely solved at each optimization iteration using continuation on Reynolds number, i.e., starting with Re = 0 and incrementing by ΔRe until Re* is reached. There is another alternative that is intermediate between the extremes of GRG (completely solving the flow equations at each iteration, using continuation on Reynolds number) and SQP (solving a linearized approximation only). This is to use the continuation at the level of the optimization problem, as described at the end of Section 3, in conjunction with full solution of the flow equations *for the current value of*

[5] Recall that with GRG the constraints are eliminated and therefore the problem is an unconstrained one.

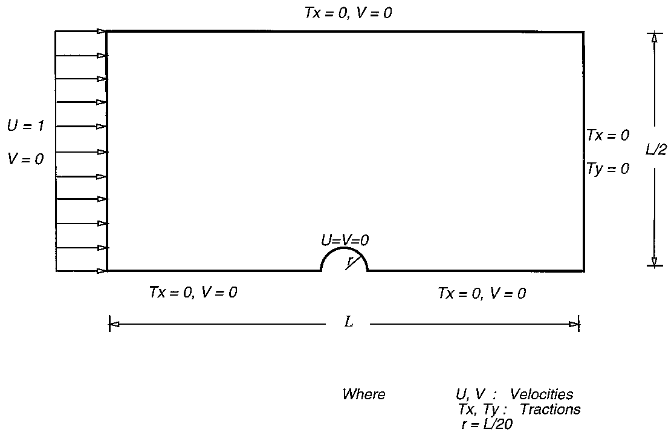

FIG. 4.3. Computational domain and boundary conditions, two-dimensional flow around infinite cylinder.

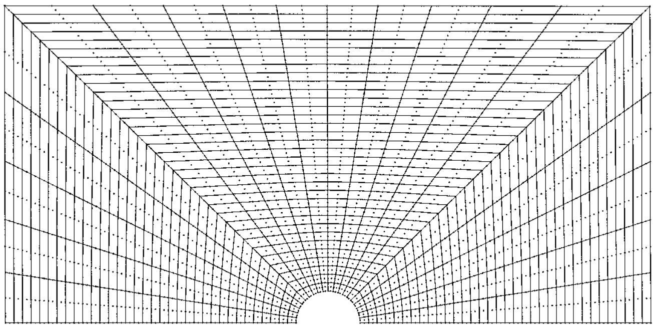

FIG. 4.4. Mesh of 680 biquadratic elements, 2829 nodes.

Re. We refer to this method as continuation QN-GRG, or CQN-GRG. It uses the converged flow solution of the previous optimization iteration as an initial guess to the velocity field of the current iteration. We expect its efficiency to be between SQP and GRG.

Finite element approximation of the continuous problem is achieved with isoparametric biquadratic rectangles in 2D and triquadratic hexahedra in 3D. These elements produce errors in the derivatives of $\mathbf{u}^h$ of $\mathcal{O}(h^2 + \varepsilon)$ [11]. All integrals are evaluated with Gauss–Legendre numerical integration using a 3×3 ($\times 3$) scheme, with the exception of the penalized terms, which are "underintegrated" with a 2×2 ($\times 2$) scheme to avoid "locking" [17]. The value of the penalty parameter ε is taken to be 10^{-7}. The flow solver has been verified against a standard benchmark, the driven cavity problem. We have chosen a value of 10^{-7} in the Euclidean norm of the first order optimality condition (3.2) to terminate optimization iterations. The Reynolds number step size for continuation, ΔRe, is in all cases 50.

Solution of systems involving $\mathbf{A}_{ss}$ and its transpose is at the heart of all five methods. In GRG, these systems characterize a Newton step on the state equations, as well as the sensitivity equations. In SQP, their presence reflects the choice of a block elimination (or a null space basis) that favors "inverting" $\mathbf{A}_{ss}$. The cost of solving these systems asymptotically dominates an optimization iteration, whether in SQP or GRG guise, since it is the only step that is superlinear in n_s (all others are linear at worst). Clearly one would like to perform these linear solves as cheaply as possible. Our initial desire was to use a Krylov subspace method, specifically the quasi-minimum residual (QMR) method, to solve the systems involving $\mathbf{A}_{ss}$ and its transpose, since methods of this type are representative of large-scale CFD solvers. However, after trying QMR on the discrete penalty-based Navier–Stokes equations, we concluded that the equations were too ill-conditioned for iterative solution to be competitive. Even incomplete LU preconditioning was ineffective in allowing convergence in reasonable time. This no doubt stems from the penalty formulation, and we expect that a different conclusion would have been reached had a mixed formulation (one that included both velocity and pressure) been chosen.

Instead, we have chosen the multifrontal sparse LU factorization code UMFPACK [4, 5] for solving the systems involving $\mathbf{A}_{ss}$ and its transpose. UMFPACK provides a routine for computing the LU factors of a given sparse matrix. Once this has been computed, UMFPACK provides further routines for finding the solutions to systems involving the triangular factors of a matrix as well as their transposes. Thus, using UMFPACK, only a single factorization of $\mathbf{A}_{ss}$ is required at each iteration of the Newton-SQP and quasi-Newton SQP methods; the primary difference between the two methods therefore lies in the number of triangular solves each performs (in addition to the computation of second derivatives). Using a sparse direct method of course ultimately limits the maximum size of problems we can solve, relative to no-fill methods such as ILU-QMR. Even though we have found UMFPACK to

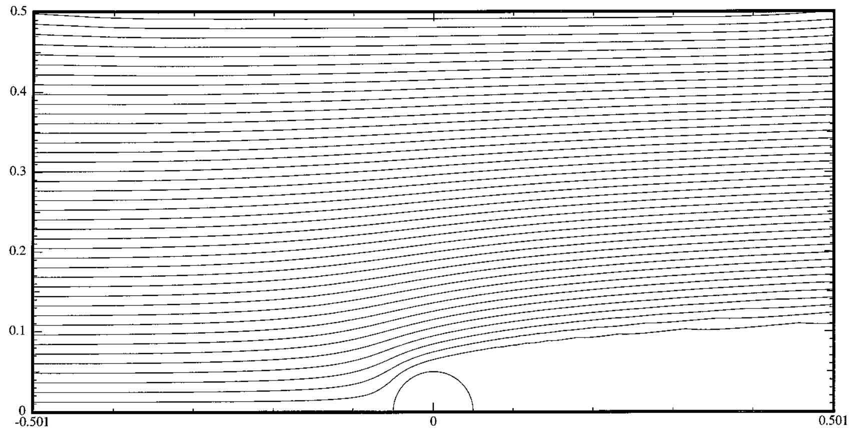

FIG. 4.5. Streamlines for steady flow around a cylinder, no control, Re = 500.

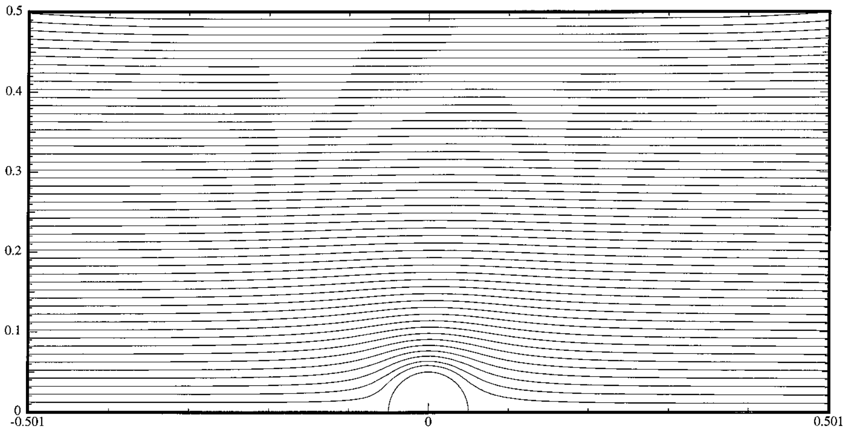

FIG. 4.6. Streamlines for steady flow around a cylinder, optimal control, Re = 500.

be very effective at reducing fill, a significant amount of fill is unavoidable for three-dimensional, higher-order, vector finite element problems, due to the high average degree of nodes in the finite element graph.

To compare the five different optimal control methods, we choose a model problem of two-dimensional flow around an infinite cylinder. Here, we define the Reynolds number as

$$\mathrm{Re} = \frac{\rho D\,|\mathbf{u}_\infty|}{\mu},$$

where D is the cylinder diameter. Without boundary control, the behavior of the velocity field with increasing Reynolds number is depicted in, for example, [1]. Flow separation is evident for Reynolds numbers as low as 10. The flowfield remains stationary and exhibits two symmetric standing eddies up to around Re = 50. Beyond this range, the wake becomes increasingly unstable and oscillatory, and a vortex street forms in the wake and persists downstream. Beyond a Reynolds number of about 60, the flow is neither symmetric about the cylinder centerline, nor steady, as assumed by our model.

In this section we solve optimal control problems for flows around a cylinder with Reynolds numbers as high as 500. However, we use the steady form of the governing flow equations as constraints; furthermore, our flow model assumes symmetry of the velocity field about the centerline of the cylinder. In principle, the optimal control problem for flow around a cylinder at Re = 500 should be constrained by the time-dependent version of the Navier–Stokes equations and should not assume symmetry of the velocity field. However, if one makes two *ad hoc* assumptions—that the optimal velocity field for the time-dependent problem is both steady and symmetric, even if the uncontrolled flow is neither—then one ought to be able to use the steady Navier–Stokes equations as constraints in the formulation of the optimal control problem and cut the computational domain in half to exploit symmetry. Computational complexity is greatly reduced under these two assumptions. It is clear that a sufficiently close initial guess to this steady, symmetric optimal control problem would converge to the optimum defined by the unsteady, unsymmetric control problem.

How can these two *ad hoc* assumptions be verified? In general, the only way is to solve the time-dependent opti-

TABLE 4.1

Number of Optimization Iterations Taken by GRG and SQP Methods

Re	SD-GRG	QN-GRG	CQN-GRG	QN-SQP	N-SQP
100	643	23	34	29	13
200	265	30	54	30	14
300	291	35	70	37	15
400	×	45	89	45	18
500	×	×	102	52	20

TABLE 4.2

Timings in Minutes for GRG and SQP Methods on a DEC 3000/700 with 225 MHz Alpha Processor and 512 Mb Memory

Re	SD-GRG	QN-GRG	CQN-GRG	QN-SQP	N-SQP
100	3766.75	74.10	41.17	18.37	16.68
200	2922.52	168.47	66.18	19.32	17.95
300	4744.93	278.98	86.77	25.02	22.10
400	×	462.65	110.52	30.72	26.27
500	×	×	126.98	35.15	30.10

mal control problem (algorithms for which we have not considered nor implemented), and compare to the optimal controls computed under the two assumptions. Alternatively, a reasonable heuristic would be: (i) compute the optimal controls using a steady model throughout the optimization; (ii) apply the resulting optimal controls as boundary conditions in a time-dependent Navier–Stokes solver; and (iii) conclude that, if the resulting velocity field is steady and symmetric, the two assumptions are likely valid.

Applying this heuristic, we use a time-dependent Navier–Stokes code to simulate flow around the cylinder in the entire domain at Re = 500. Figure 4.1 shows a snapshot of fluid streamlines at $t = 3.5$ s for the uncontrolled case (i.e., no-slip boundary conditions on the cylinder surface); clearly the flow is unsymmetric about the horizontal axis; furthermore, integration in time reveals no steady state. However, if we apply the steady optimal controls (i.e., those found by solving the steady optimal control problem) as boundary conditions at nine equally spaced points on the backside of cylinder, and solve using the time-dependent Navier–Stokes code, we obtain the streamlines shown in Fig. 4.2. The streamlines are indeed symmetric, and further integration in time does not show a change in the velocity field. In fact, using a steady Navier–Stokes code we obtain the same velocity field as in Fig. 4.2.

Therefore, while the initial, uncontrolled flow is unsymmetric and unsteady, the optimal (with respect to a steady model) flow is both steady and symmetric (from the point of view of a time-dependent solver) for flow around a cylinder at Re = 500. This motivates using the (steady) flow model of Section 2, and considering only one-half of the flow domain, thereby reducing the size of the forward problem. We stress that we have not *proven* that there does not exist an optimal control computed using an unsteady model that has a lower value of the dissipation function than that of the steady optimal control problem. However, this seems unlikely, and it is clear that the steady optimal control is at least a local optimum for the unsteady problem of Re = 500 flow around a cylinder.

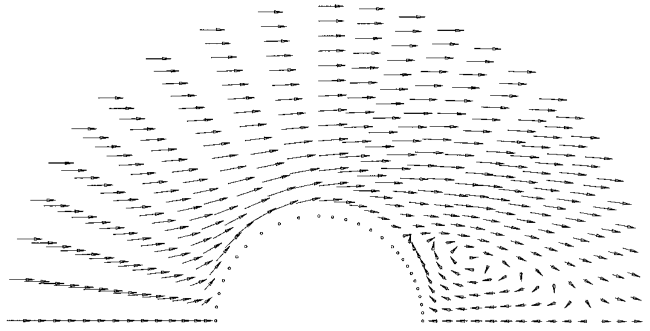

FIG. 4.7. Velocity field for steady flow around a sphere, no control, Re = 130.

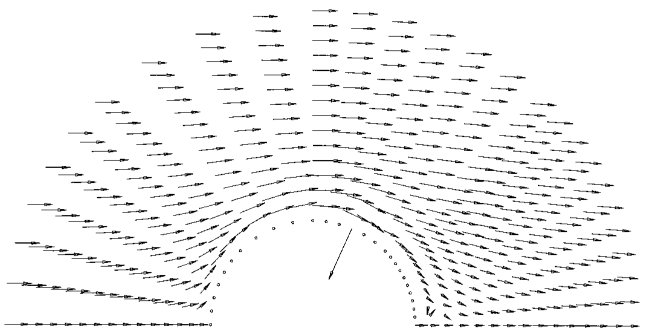

FIG. 4.8. Velocity field for steady flow around a sphere, Case 1 optimal control, Re = 130.

Thus, we consider the computational domain and boundary conditions depicted in Fig. 4.3 and associated mesh of Fig. 4.4. The mesh uses 680 isoparametric biquadratic elements, resulting in $N = 2829$ nodes and $n = 5658$ unknown velocity components. Boundary control of the velocity is applied at five equally spaced points on the backside of the cylinder. Since each has two velocity components, we have a total of $n_c = 10$ control variables. Streamlines for the case of no control and Re = 500 are shown in Fig. 4.5.[6] The streamlines are seen to detach near the top of the cylinder, and there is a large recirculation zone behind the cylinder. After solving the optimization problem (2.12)–(2.13), the streamlines shown in Fig. 4.6 are obtained. The resulting flow resembles a potential flow, and separation is greatly reduced.

As a comparison of the methods, we solve a sequence of five optimization problems, corresponding to Re = 100, 200, 300, 400, and 500, using the five optimization methods described above. Table 4.1 compares the number of optimization iterations taken by the five methods. For the continuation methods, i.e., CQN-GRG, QN-SQP, and N-SQP, the numbers reported in the table are the sum of iterations across all optimization problems (each corresponding to a value of Re).

As expected, SD-GRG takes by far the largest number of iterations. The symbol "×" means that the method failed to converge to a stationary point, which occurred with SD-GRG for Re = 400 and 500, and QN-GRG for Re = 500. QN-GRG takes an order of magnitude fewer iterations than SD-GRG, due to its ability to approximate curvature of the control variable space. However, CQN-GRG takes almost twice as many iterations as QN-GRG. The reason is that the sequence of Reynolds number steps is "packed" into a single optimization iteration with QN-GRG, while CQN-GRG "promotes" the continuation on Re to the level of the optimization problem; thus, these steps contribute to the number reported in Table 4.1. On

[6] Of course, this is not a physically meaningful flow.

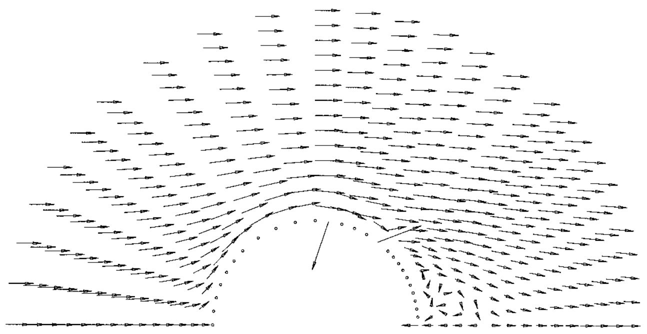

FIG. 4.9. Velocity field for steady flow around a sphere, Case 2 optimal control, Re = 130.

the other hand, the cost per iteration of CQN-GRG should be significantly lower than QN-GRG, since each flow solution need only be converged for the current Reynolds number, and the solver benefits from an initial guess taken from the previous converged value. QN-SQP reduces the number of iterations by almost 50% over CQN-GRG, since it liberates the optimizer from having to follow a path dictated by satisfaction of the flow equations. The result is that the number of iterations taken by QN-GRG and QN-SQP are similar. Of course the cost per iteration of QN-GRG (and of CQN-GRG) will be much higher than QN-SQP, which will be reflected in CPU time. N-SQP offers the best performance from the point of view of iterations taken, providing on average 2.5 times fewer iterations than the QN methods. Of course, this reduction in steps taken must be balanced with increased work per iteration associated with N-SQP relative to QN-SQP.

Table 4.2 shows timings of each method for the sequence of Reynolds numbers solved. Note that these timings are in minutes, so the SD method requires several days to find an optimal solution, which is unacceptable. The QN-GRG method offers over an order of magnitude reduction, but the measured times are still on the order of hours. Partially integrating flow solution with optimization, as in CQN-GRG, further reduces CPU time by a factor of about three. A further factor of three reduction is achieved by fully integrating flow solution with optimization, through the use of QN-SQP. This results from its requiring only two linear solves per iteration, against CQN-GRG's fully converging the flow equations. On the other hand, CPU time decreases only marginally when the N-SQP method is used, typically between 10 and 15%, even though the number of iterations is significantly lower. This results from the additional work N-SQP must do at each iteration, chiefly through the additional right-hand side solves and the construction of the exact reduced Hessian $\mathbf{W}_z$ in (3.6). While the cost of constructing $\mathbf{W}_z$ is linear in n_s, the constant is large, since it involves element generation- and assembly-like finite element computations. The conclusion is that, while the *asymptotic* costs per iteration of QN-GRG and N-SQP are the same,[7] it turns out that, for the value of n_s we are considering, the lower order terms contribute meaningfully, and conspire to make N-SQP roughly twice as expensive per iteration as QN-SQP. Still, the Newton method does take less time; whether this is worth the additional effort of implementing second derivatives will depend on the particular application.

Finally, we demonstrate the capability of the reduced Hessian SQP optimal control method by solving a three-dimensional optimal flow control problem. We select a model problem of flow around a sphere at a Reynolds number of 130, which is the limit of steady flow for this problem [1]. Again, we exploit symmetry, this time about two orthogonal planes, to obtain a quarter model of the sphere. Along the planes of symmetry, zero tangential traction and zero normal velocity are specified as boundary conditions. A spherical far-field boundary truncates the flow domain, and on it are imposed traction-free boundary conditions. The computational domain is meshed into 455 isoparametric triquadratic prism elements, resulting in $N = 4{,}155$ nodes and $n = 12{,}465$ unknown velocity components. In the absence of boundary velocity control, the flow separates and forms a standing ring-eddy behind the sphere, as shown in Fig. 4.7.

We introduce boundary velocity control at distinct points lying on six planes aligned with the direction of the flow, and perpendicular to the surface of the sphere. We consider three cases consisting of one, three, and five holes on each plane. These correspond to 6, 13, and 25 boundary holes.[8] The total number of control variables is 16, 33, and 65 for Cases 1, 2, and 3, respectively.[9] The location of the holes is evident from Figs. 4.8–4.10. Because of the expense associated with solving three dimensional flow control problems, we solve the optimization problem (2.12)–(2.13) using only the most efficient of the methods considered here, i.e., the Newton-SQP method. The optimal flow field on a vertical plane of symmetry aligned with the flow is shown for Case 1 in Fig. 4.8. The figure indicates the magnitude and direction of the control velocity at the single boundary point appearing on this plane. The application of suction at six such points has kept the streamlines from separating from the sphere, with the result that the standing ring eddy is largely eliminated. Consider now Case 2, with twice as many control variables. Figure 4.9 shows the velocity field corresponding to the optimal control. While this flowfield appears to be an improvement over the uncontrolled flow (Fig. 4.7), there is some recirculation immediately downstream of the sphere that was not present in

[7] When sparse LU is used to factor $\mathbf{A}_{ss}$ and under the assumption $n_s \gg n_c$.

[8] In Cases 2 and 3, there is a shared hole among the six planes.

[9] Some velocity components are required to be zero due to symmetry.

TABLE 4.3

Results of Optimal Control of Flow around a Sphere

	No control	Case 1	Case 2	Case 3
No. of control variables	—	16	33	65
Optimal dissipation	5.775457	1.729944	1.551276	1.430053

the Case 1 optimal solution. Indeed, we expect that the Case 2 optimal solution should be at least as good as Case 1, since the former duplicates the holes of the latter while adding several more. However, the optimizer "sees" only the value of the objective function (and its derivatives), and in Case 2 the optimal value of the dissipation function is indeed lower than that of Case 1, as seen in Table 4.3. The Case 3 optimal flowfield, shown in Fig. 4.10, resembles Case 2, but as seen in Table 4.3, its dissipation function is lower than that of Case 2, so the result matches our expectation.[10]

Because of the large computational burden imposed by the three-dimensional problems, we did not conduct careful timings. However, the wall-clock time required (on the workstation described in Table 4.2) in all three cases was less than a day. The majority of the time undoubtedly was spent on factoring the coefficient matrix $\mathbf{A}_{ss}$.

5. FINAL REMARKS

Based on the comparison of the previous section, we conclude that the reduced Hessian SQP methods are overwhelmingly superior to the GRG methods that are popular for optimization problems involving PDEs as constraints, offering over an order of magnitude improvement in time required for the optimal flow control problems considered. In particular, the methods are so efficient that the optimal control for a two-dimensional flow around a cylinder at Reynolds number 500 is found in about a half hour on a desktop workstation. Furthermore, the Newton-SQP method is capable of solving some three-dimensional optimal flow control problems on the same workstation in less than a day. Even though the Newton-SQP method takes significantly fewer iterations, its need to construct exact Hessians and to solve linear systems that number on the order of the number of control variables makes it only marginally more efficient than its quasi-Newton counterpart, based on our two-dimensional results.

The GRG methods described here represent a strict interpretation of the GRG idea—at each optimization iteration, the flow equations are converged to a tolerance of 10^{-7}. However, in the spirit of SQP, one might choose a greater value of the tolerance at early iterations, making sure to reduce the tolerance to its target value as the optimization iterations converge. Indeed, in the limit of one Newton step on the flow equations per optimization iteration, we essentially recover the reduced SQP method. A related idea is to pose the early optimization iterations on a coarse mesh, and refine as the optimum is approached; this can be very effective, as advanced in [16] and [27]. Finally, in [3], a number of possibilities are defined that are intermediate between the extremes of GRG (full flow convergence per optimization iteration) and reduced SQP (one state equation Newton step per optimization iteration).

We imagine that for larger problems, the cost of factoring the state equation Jacobian matrix will begin to display its asymptotic behavior and dominate the lower-order terms, leading to increasing efficiency of the Newton method relative to quasi-Newton. However, problem size cannot continue to grow indefinitely and still allow sparse LU factorization. For example, in solving three dimensional Navier-Stokes flow control problems on the workstation described in Table 4.2, we encountered a limit of about 13,000 state variables, due to memory. Beyond this size, where undoubtedly most industrial-scale flow control problems lie, iterative solvers are required, and it remains to be seen whether they can be tailored to multiple righthand sides sufficiently well that Newton SQP can retain its superiority over quasi-Newton SQP.

For the largest problems, parallel computing will become essential. Of course, there is a long history of parallel algorithms and implementations for the forward problem, i.e., Navier–Stokes flow simulation. Algorithms 3.1 and 3.2 are well suited to parallel machines, since the majority of their work involves solution of linear systems having state equation Jacobians as their coefficient matrix; this is just a step of the forward problem, the parallelization of which is well-understood. Indeed, in [9] we discuss the parallel implementation of Algorithm 3.2 for a problem in shape optimization governed by compressible flows.

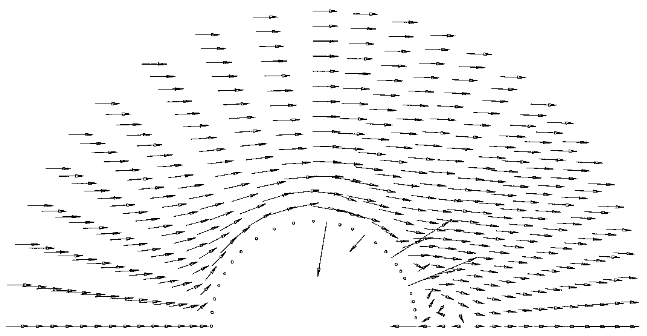

FIG. 4.10. Velocity field for steady flow around a sphere, Case 3 optimal control, Re = 130.

[10] Even if this were not the case, the issue of local vs global optimality might explain such an apparently anomalous result.

ACKNOWLEDGMENTS

We thank Beichang He for providing the time dependent Navier–Stokes simulations that were used to verify the symmetry and stationarity of the optimally controlled flow. We also thank Cristina Amon, Carlos Orozco, Larry Biegler, and James Antaki for various discussions and suggestions related to this work.

REFERENCES

1. G. K. Batchelor, *An Introduction to Fluid Dynamics* (Cambridge Univ. Press, Cambridge, UK, 1967).
2. L. T. Biegler, J. Nocedal, and C. Schmid, A reduced Hessian method for large-scale constrained optimization, *SIAM. J. Optim.* **5,** 314 (1995).
3. E. J. Cramer, J. E. Dennis, P. D. Frank, R. M. Lewis, and G. R. Shubin, Problem formulation for multidisciplinary optimization, *SIAM J. Optim.* **4,** 754 (1994).
4. T. A. Davis, *Users' Guide for the Unsymmetric Pattern Multifrontal Package (UMFPACK),* Tech. Rep. TR-93-020, CIS Dept., University of Florida, Gainesville, FL, 1993.
5. T. A. Davis and I. S. Duff, *An Unsymmetric Pattern Multifrontal Method for Sparse LU Factorization,* Tech. Rep. TR-93-018, CIS Dept., University of Florida, Gainesville, FL, 1993.
6. C. Farhat and P.-S. Chen, Tailoring domain decomposition methods for efficient parallel coarse grid solution and for systems with many right hand sides, in *Domain Decomposition Methods in Science and Engineering, Contemporary Mathematics,* Vol. 180 (American Mathematical Society, Providence, RI), 1994.
7. D. Gabay, Reduced Quasi-Newton methods with feasibility improvement for nonlinearly constrained optimization, *Math. Programming Study* **16,** 18 (1982).
8. M. Gad-el-Hak, Flow control, *Appl. Mech. Rev.* **42,** 261 (1989).
9. O. Ghattas and C. E. Orozco, A parallel reduced Hessian SQP method for shape optimization, in *Multidisciplinary Design Optimization: State-of-the-Art,* edited by N. Alexandrov and M. Hussaini (SIAM, Philadelphia, 1997), p. 133.
10. P. E. Gill, W. Murray, and M. H. Wright, *Practical Optimization* (Academic Press, New York, 1981).
11. M. D. Gunzburger, *Finite Element Methods for Viscous Incompressible Flows* (Academic Press, San Diego, 1989).
12. M. D. Gunzburger, ed., *Flow Control,* IMA Volumes in Mathematics and Its Applications, Vol. 68. (Springer-Verlag, Berlin/New York, 1995).
13. M. D. Gunzburger, L. S. Hou, and T. P. Svobodny, Optimal control and optimization of viscous, incompressible flows, in *Incompressible Computational Fluid Dynamics,* edited by M. D. Gunzburger and R. A. Nicolaides (Cambridge Univ. Press, Cambridge, UK, 1993), Ch. 5, p. 109.
14. R. T. Haftka, Z. Gürdal, and M. P. Kamat, *Elements of Structural Optimization* (Kluwer Academic, Dordrecht/Norwall, MA, 1990).
15. E. J. Haug and J. S. Arora, *Applied Optimal Design* (Wiley–Interscience, New York, 1979).
16. W. P. Huffman, R. G. Melvin, D. P. Young, F. T. Johnson, J. E. Bussoletti, M. B. Bieterman, and C. L. Hilmes, Practical design and optimization in computational fluid dynamics, in *Proceedings of 24th AIAA Fluid Dynamics Conference, Orlando, Florida, July 1993.*
17. T. J. R. Hughes, W. K. Liu, and A. Brooks, Finite element analysis of incompressible viscous flow by the penalty function formulation, *J. Comput. Phys.* **30,** 1 (1979).
18. M. S. Khaira, G. L. Miller, and T. J. Sheffler, *Nested Dissection: A Survey and Comparison of Various Nested Dissection Algorithms,* Tech. Rep. CMU-CS-92-106R, Carnegie Mellon University, 1992.
19. F. S. Kupfer and E. W. Sachs, A prospective look at SQP methods for semilinear parabolic control problems, in *Optimal Control of Partial Differential Equations,* edited by K. Hoffman and W. Krabs (Springer-Verlag, Berlin/New York, 1991), p. 145.
20. P. Moin and T. Bewley, Feedback control of turbulence, *App. Mech. Rev.* **47,** S3 (1994).
21. C. E. Orozco and O. Ghattas, Massively parallel aerodynamic shape optimization, *Comput. Syst. Eng.* **1–4,** 311 (1992).
22. C. E. Orozco and O. Ghattas, Optimal design of systems governed by nonlinear partial differential equations, in *Fourth AIAA/USAF/NASA/OAI Symposium on Multidisciplinary Analysis and Optimization, AIAA, 1992,* p. 1126.
23. L. Prandtl and O. G. Tietjens, *Applied Hydro- and Aeromechanics* (Dover, New York, 1934), p. 81.
24. U. Ringertz, *Optimal Design of Nonlinear Shell Structures,* Tech. Rep. FFA TN 91-18, The Aeronautical Research Institute of Sweden, 1991.
25. U. Ringertz, An algorithm for optimization of nonlinear shell structures, *Int. J. Numer. Methods Eng.* **38,** 299 (1995).
26. Y. Xie, *Reduced Hessian Algorithms for Solving Large-Scale Equality Constrained Optimization Problems,* Ph.D. thesis, University of Colorado, Boulder, Department of Computer Science, 1991.
27. D. P. Young, W. P. Huffman, R. G. Melvin, M. B. Bieterman, C. L. Hilmes, and F. T. Johnson, Inexactness and global convergence in design optimization, in *Proceedings of the 5th AIAA/NASA/USAF/ISSMO Symposium on Multidisciplinary Analysis and Optimization, Panama City, Florida, September 1994.*