Technische Universität Berlin
Institut für Mathematik

Adjoint gradients compared to gradients from algorithmic differentiation in instantaneous control of the Navier-Stokes equations

Michael Hinze (TU Dresden) and Thomas Slawig

Preprint 0735-02

Preprint-Reihe des Instituts für Mathematik
Technische Universität Berlin

Preprint 0735-02 April 2002

Abstract

Gradients computed via an adjoint equation and obtained algorithmically by Automatic Differentiation (AD) tools are compared with respect to accuracy and performance. As model application the method of instantaneous control for the time-dependent incompressible Navier-Stokes equations with distributed control is used. The method of instantaneous control and both approaches to obtain the gradient are described in detail. Implementation issues of the AD process are given. Numerical results for gradient evaluations and complete control runs on different grids are presented.

Adjoint gradients compared to gradients from algorithmic differentiation in instantaneous control of the Navier-Stokes equations

Michael Hinze (TU Dresden) and Thomas Slawig

April 10, 2002

1 Introduction

In this work we compare accuracy and performance of adjoint gradients (i.e. computed by the discretization of an adjoint equation) with gradients obtained from algorithmic or automatic differentiation (AD). As model problem we consider instantaneous control for the instationary Navier-Stokes system with distributed controls in two spatial dimensions. The paper focusses on the comparison of the generation of gradients used in iterative optimization algorithms to solve control problems. We discuss gradients computed by the solution of a discretized adjoint equation, and those generated algorithmically from the discretized cost functional. One goal of this work is to investigate whether discretization and differentiation for this special control problem commute, at least in the sense that the results are comparable up to the discretization error. Moreover we are interested in a comparison of performance.

AD is the most important tool for generating derivatives of functions represented by computer programs. It therefore is one of the key technologies for the step from *model based numerical simulation* to *model based design with numerical methods*. It is one goal of this paper to illustrate the use of modern AD tools for providing derivatives in comparison to the hand coded approach. As model application we choose the instantaneous control method which is described in detail in Section 2.

In the next section we describe a general class of control problems for the incompressible Navier-Stokes equations. Thereafter we present the basic ideas of instantaneous control. The two fundamentally different approaches of computing gradients via an adjoint equation or via AD are discussed in the next two sections. In Section 5 we focus on the preparation details of applying AD software to the Navier-Stokes solver. This includes the effective differentiation of iterative schemes such as conjugate gradient methods. Finally we present numerical results to compare accuracy and efficiency of the two approaches to generate gradients that can be used to solve control problems.

2 Instantaneous Control

To explain the method of instantaneous control we start with writing the Navier-Stokes system in sinoidal setting: Find a velocity field $x : (0,T) \times \Omega \to \mathbb{R}^2$ such that

$$\dot{x} + Ax + n(x) = 0 \text{ in } \Omega^T := (0,T) \times \Omega, \quad x(0) = x_0 \text{ in } \Omega, \tag{2.1}$$

where $\Omega \subset \mathbb{R}^2$ denotes the bounded spatial domain, A is the Stokes operator with domain of definition $D(A) = H^2(\Omega)^2 \cap V$, i.e. A satisfies homogeneous Dirichlet boundary conditions, and $n(y) := (y\nabla)y$ denotes the nonlinearity. Here the initial values are assumed in $H := clos_{L^2}\left(\{v \in C_0^\infty(\Omega)^2 : \operatorname{div} v = 0\}\right)$, and $V := clos_{H^1}\left(\{v \in C_0^\infty(\Omega)^2 : \operatorname{div} v = 0\}\right)$ denotes the solinoidal vector fields in $H^1(\Omega)^2$, see [5] for the proper analytical setting.

The approach taken is based on a time discretization of equation (2.1). For this purpose let $0 = t_0 < t_1 < \cdots < t_m = T$ denote an equidistant grid on the time interval $[0, T]$ with step size $\tau = \frac{T}{m}$. At each discrete time level t_i a stationary control problem is solved for an approximate optimal control u_i^*. This control is used to steer the system from t_i to t_{i+1} towards a given function z, where a new approximate optimal control is determined. Unless otherwise specified from now onwards z denotes a sufficiently smooth bounded time dependent function.

As time discretization method the implicit Euler method is chosen, and as cost function we use

$$J(u) \quad := \quad \hat{J}(x,u) \;=\; \frac{\gamma}{2}|u|^2_{L^2(\Omega)^2} \;+\; \frac{1}{2}|x - z|^2_{L^2(\Omega)^2} \tag{2.2}$$

with $u \in U := H$, i.e. we choose H as control space. Note, that a more general setting is also possible, see [9]. The optimization problem in every time step then has the form

$$(P^j) \qquad \begin{cases} \min\limits_{u^{j+1} \in U} J(u^{j+1}) = \dfrac{\gamma}{2}|u^{j+1}|^2_{L^2(\Omega)^2} \;+\; \dfrac{1}{2}|x^{j+1} - z^j|^2_{L^2(\Omega)^2} \\ \text{s.t.} \qquad (I + \tau A)x^{j+1} \;=\; x^j + \tau b^j + u^{j+1} \quad \text{in } V, \end{cases}$$

which, due to the quadratic character of the cost function, admits a unique solution (x^{j+1}, u^{j+1}). Note that the subsidiary condition in (P^j) stems not from the discretization of a continuous dynamical system, i.e. there is no τ in front of u^{j+1} missing.

We denote by $D_u J(u)v$ the directional derivative of J at u in direction $v \in U$, and by $J'(u)$ the Riesz representative of the operator $v \mapsto D_u J(u)v$ in U, i.e. the unique element satisfying

$$D_u(u)v \quad = \quad (J'(u), v) \qquad \text{for all } v \in U$$

where $(\cdot, \cdot)$ denotes the inner product in $L^2(\Omega)^2$.

Now, at every time instance t_i we apply exactly one step of the gradient algorithm to approximately solve (P^j). This approach from now on will be referred to as **instantaneous control**. It may be interpreted as an in-exact variant of finite-horizon model predictive control [6, 15]. The method has succesfully been applied to compute suboptimal control strategies for fluid flows [3, 4, 11] and recently, also to the cooling of steel [16]. The stabilizing properties of the method in control of the Burgers equation are investigated in [10], for the Navier-Stokes equations see [9, Ch. 5]. In algorithmical form it may be rewritten as follows.

Algorithm 2.1. (Instantaneous control)

1. Given initial values x^0, set $j = 0$, $t_0 = 0$
2. Given u_0^j, compute $g := J'(u_0^j)$.
3. Given ρ, set $u^{j+1} = u_0^j - \rho g$ and solve
$$(I + \tau A)x^{j+1} = x^j + \tau b^j + u^{j+1}$$
4. Set $t_{j+1} = t_j + \tau$, $j = j + 1$
5. If $t_j < T$ goto 2.

Note that the optimal step size $\rho = \rho^*$ in step 4 of Algorithm 2.1 can be computed exactly and is given by

$$\rho^* = - \quad \frac{(J'(u), g)}{\gamma |g|^2_{L^2(\Omega)^2} + |x(g)|^2_{L^2(\Omega)^2}}, \tag{2.3}$$

where for a given direction d, $x(d)$ is the state computed from

$$(I + \tau A)x = x_o + \tau b + d \text{ in } V.$$

Here, x_o denotes the state at the previous time instance.

Problem (P^j), the method of instantaneous control and the above algorithm can also be formulated in a finite-dimensional setting: Since we are dealing with distributed control for simplicity we introduce a common finite-dimensional space V_h as approximations of both V and U. This choice further is motivated by the relation of controls and adjoint states, compare (3.1). The third equation there suggests to use the same discretization space for both control variables and adjoint state. Here we discuss only finite-element spaces V_h, but all further considerations will apply for different discretization schemes in a rather similar way.

After discretization we obtain the optimization problem

$$(P_h^j) \qquad \begin{cases} \min\limits_{u_h^{j+1} \in V_h} J_h(u_h^{j+1}) = \dfrac{\gamma}{2}|u_h^{j+1}|^2_{L^2(\Omega)^2} + \dfrac{1}{2}|x_h^{j+1} - z_h^j|^2_{L^2(\Omega)^2} \\ \text{s.t.} \qquad (I_h + \tau A_h)x_h^{j+1} = x_h^j + \tau b_h^j + u_h^{j+1} \text{ in } V_h \end{cases}$$

Algorithm 2.1 retains its structure with all quantities replaced by finite-dimensional counterparts (denoted by the subscript h). Step 2 becomes

2. Given u_{0h}^j, compute $g_h := F_h(u_{0h}^j)$.

It has to be specified what F_h means, i.e. how g_h is computed. One approach is to first derive a representation for the derivative J' of the continuous functional J in (P^j) and then to replace the continuous ingredients of J' by their discrete analogues. This is called *first optimize, then discretize.* The other is based on Algorithmic Differentiation of a discrete version of the cost functional and is called *first discretize then optimize.* To compare both approaches we thus have to introduce a discrete cost functional. For this purpose we define the mapping

$$\begin{aligned} \mathcal{I}_h \quad : \quad \mathbb{R}^n \quad &\to \quad V_h \;=\; \text{span}\{\phi_i, i = 1, \ldots, n\}, \\ \mathtt{v} = (\mathtt{v}_i)_{i=1,\ldots,n} \quad &\mapsto \quad v_h \;=\; \sum_{i=1}^{n} \mathtt{v}_i \phi_i, \end{aligned}$$

where ϕ_i are the finite element basis functions. Now we transform (P_h^j) onto the coefficient space $\mathbb{R}^n$. If $V_h \subset V$ we may just set $J_h = J$, otherwise an approximation has to be defined. For given $u_h = \mathcal{I}_h \mathtt{u}, x_h = \mathcal{I}_h \mathtt{x}$, and $z_h = \mathcal{I}_h \mathtt{z}$ we now define

$$\begin{aligned} \mathtt{J}(\mathtt{u}) \;:=\; J_h(u_h) \;=\; \hat{J}_h(u_h, x_h) \quad &= \quad \hat{J}(\mathcal{I}_h \mathtt{u}, \mathcal{I}_h \mathtt{x}) \\ &= \quad \frac{\gamma}{2}(\mathcal{I}_h \mathtt{u}, \mathcal{I}_h \mathtt{u}) + \frac{1}{2}(\mathcal{I}_h(\mathtt{x} - \mathtt{z}), \mathcal{I}_h(\mathtt{x} - \mathtt{z})) \\ &= \quad \frac{\gamma}{2}\mathtt{u}^T \mathtt{M} \mathtt{u} + \frac{1}{2}(\mathtt{x} - \mathtt{z})^T \mathtt{M}(\mathtt{x} - \mathtt{z}) \end{aligned}$$

where $\mathtt{M}$ is the finite element mass matrix. The state equation becomes

$$(\mathtt{M} + \tau \mathtt{A})\mathtt{x}^{j+1} \quad = \quad \mathtt{M}(\mathtt{x}^j + \tau \mathtt{b}^j + \mathtt{u}^{j+1}) \quad \text{in } \mathbb{R}^n \tag{2.4}$$

where $\mathtt{A}$ is the finite element stiffness matrix in the solinoidal velocity space. Now we obtain the discrete problem

$$(\mathtt{P}^j) \qquad \min_{\mathtt{u}^{j+1} \in \mathbb{R}^n} \mathtt{J}(\mathtt{u}^{j+1}) \quad \text{s.t.} \quad (2.4)$$

The discrete version of Algorithm 2.1 now reads

Algorithm 2.2. (Instantaneous control, discrete version)

1. Given initial values $\mathtt{x}^0$, set $j = 0$, $t_0 = 0$
2. Given $\mathtt{u}_0^j$, compute $\mathtt{g}$ such that $\mathcal{I}_h \mathtt{g} = g_h$.

3. Given ρ, set $\mathtt{u}^{j+1} = \mathtt{u}_0^j - \rho\mathtt{g}$ and solve

$$(\mathtt{M} + \tau\mathtt{A})\mathtt{x}^{j+1} = \mathtt{M}(\mathtt{x}^j + \tau\mathtt{b}^j + \mathtt{u}^{j+1})$$

4. Set $t_{j+1} = t_j + \tau$, $j = j + 1$

5. If $t_j < T$ goto 2.

The two alternative ways to compute $\mathtt{g}$ in step 2 are presented in the next two sections.

3 Computation of the Gradient via the Adjoint Equation

The solution pair (x^{j+1}, u^{j+1}) of problem (P^j) together with the uniquely determined Lagrange multiplier μ^{j+1} solves the corresponding first order optimality conditions given by

$$\begin{array}{rcl} (I + \tau A)x^{j+1} & = & x^j + \tau b^j + u^{j+1} \text{ in } V \\ (I + \tau A^*)\mu^{j+1} & = & -(x^{j+1} - z^j) \text{ in } V \\ \gamma u^{j+1} - \mu^{j+1} & = & 0 \text{ in } L^2(\Omega^2). \end{array} \tag{3.1}$$

It is not hard to prove that this system has a unique solution, compare [9, 10]. System (3.1) can be used to obtain a representation of the derivative of the functional J at u in direction v. It is given by

$$D_u J(u)(v) \quad = \quad (J'(u), v) \; = \; (\gamma u - \mu, v) \tag{3.2}$$

and thus $g = J'(u) = \gamma u - \mu$. In Algorithm 2.1 step 2 now can be written as

2. (a) Given u_0^j, solve

$$\begin{array}{rcl} (I + \tau A)x & = & x^j + \tau b^j + u_0^j \\ (I + \tau A^*)\mu & = & -(x - z^j) \end{array}$$

   (b) Set $g := J'(u_0^j) = \gamma u_0^j - \mu$.

In the finite-dimensional space V_h all quantities again have the subscript h and we write the second part as

   (b) Set $g_h := \gamma u_{0h}^j - \mu_h$.

Thus g_h can be interpreted as $(J')_h(u_{0h}^j)$, i.e. $F_h = (J')_h$. For the discrete Algorithm 2.2 we have to find $\mathtt{g} := \mathcal{I}_h^{-1} g_h$. Since

$$\mathtt{g}^T\mathtt{M}\mathtt{v} \quad = \quad (g_h, v_h) \; = \; (\gamma u_h - \mu_h, v_h) = (\gamma\mathtt{u} - \mathtt{m})^T\mathtt{M}\mathtt{v},$$

where $\mathtt{m}, \mathtt{v}$ denote the coefficient vectors of μ_h, v_h, respectively, we have

$$\mathtt{g} \quad = \quad \gamma\mathtt{u} - \mathtt{m}.$$

Now we obtain for step (2) in Algorithm 2.2:

2. (a) Given $\mathtt{u}_0^j$, solve

$$\begin{array}{rcl} (\mathtt{M} + \tau\mathtt{A})\mathtt{x} & = & \mathtt{M}(\mathtt{x}^j + \tau\mathtt{b}^j + \mathtt{u}_0^j) \\ (\mathtt{M} + \tau\mathtt{A}^T)\mathtt{m} & = & -\mathtt{M}(\mathtt{x} - \mathtt{z}^j) \end{array}$$

   (b) Set $\mathtt{g} = \gamma\mathtt{u}_0^j - \mathtt{m}$.

An alternative approach to compute an approximation of $J'(u)$ is presented in the following section.

4 Computation of the Gradient via Algorithmic Differentiation

Algorithmic (or Automatic) Differentiation (AD) is a software technology that provides a means to compute the derivative of a function given in the form of a computer programme. Thus AD operates only on the discrete level.

Let us consider a discrete function

$$\mathtt{y} = \mathtt{J}(\mathtt{u})$$

realized in a computer programme with

- an independent (or input) variable $\mathtt{u} \in \mathbb{R}^n$
- and a dependent (or output) variable $\mathtt{y} \in \mathbb{R}^m$.

The function $\mathtt{J}$ can be represented as a concatenation

$$\mathtt{J} = \mathtt{J}_k \circ \ldots \circ \mathtt{J}_2 \circ \mathtt{J}_1$$

of k elementary intrinsic functions and operators $\mathtt{J}_i$ of the used programming language. Note that k can be very large. Each $\mathtt{J}_i$ can be differentiated *exactly* by standard rules of calculus. Using the chain rule of differentiation the derivative of the concatenated function $\mathtt{J}$ then can be written as the matrix product

$$D_{\mathtt{u}}\mathtt{y} \;=\; D_{\mathtt{u}}\mathtt{J}(\mathtt{u}) \;\;=\;\; D_{\mathtt{u}_{k-1}}\mathtt{J}_k(\mathtt{u}_{k-1}) \cdots D_{\mathtt{u}_1}\mathtt{J}_2(\mathtt{u}_1) D_{\mathtt{u}}\mathtt{J}_1(\mathtt{u}), \tag{4.1}$$

where the $\mathtt{u}_i := \mathtt{J}_i(\mathtt{u}_{i-1})$ denote intermediate (or *active*) values ($\mathtt{u}_0 = \mathtt{u}, \mathtt{u}_k = \mathtt{y}$).

There are two ways of differentiating the elementary functions and operators: In the so-called *source transformation* method of AD additional variables $\mathtt{u}_i' := D_{\mathtt{u}}\mathtt{u}_i$ are introduced and a corresponding derivative statement

$$\mathtt{u}_i' \;=\; D_{\mathtt{u}}\mathtt{u}_i \;\;=\;\; D_{\mathtt{u}_{i-1}}\mathtt{J}_i(\mathtt{u}_{i-1}) \cdot D_{\mathtt{u}}\mathtt{u}_{i-1} \;=\; D_{\mathtt{u}_{i-1}}\mathtt{J}_i(\mathtt{u}_{i-1}) \cdot \mathtt{u}_{i-1}'$$

is added for every statement

$$\mathtt{u}_i \;\;=\;\; \mathtt{J}_i(\mathtt{u}_{i-1}).$$

In this way a new source code computing both $\mathtt{y}$ and $D_{\mathtt{u}}\mathtt{y}$ is generated.

In the second approach based on *Operator Overloading (OO)* a new data type for the pairs $(\mathtt{u}_i, \mathtt{u}_i')$ of values and derivatives is introduced. A library provides the implementations of the elementary functions and operators for this data type. The *OO* approach thus does not generate new source code, but the original one is run with all active variables declared with a different type and linked with the AD library.

There are also two ways of evaluating (4.1), namely from $\mathtt{J}_1$ to $\mathtt{J}_k$ propagating the derivatives in so-called *forward mode* or backwards from $\mathtt{J}_k$ to $\mathtt{J}_1$ in *reverse mode*. As can be easily seen by analyzing the dimensions of the matrices $D_{\mathtt{u}_{i-1}}\mathtt{J}_i$ and their intermediate products in (4.1) the reverse mode saves computational cost and storage if $n < m$, i.e. if the number of output variables is significantly small compared to the number of input variables. This is the case in a control setting as presented in Section 2: Here n equals the number of control parameters which is proportional to the number of grid points in the control domain, whereas $m = 1$ due to the fact that a real-valued cost functional is minimized.

To estimate the computational effort of the reverse mode it has to be considered that either storage or recomputation (or a combination of both) of the intermediate values u_i becomes necessary. This can be easily seen in (4.1), too. For more details on these technical issues of AD see e.g.[7],[8].

The choice of the AD software moreover depends on the programming language. An overview of the available software can be found in [2]. For our application written in Fortran two tools working with source transformation were used.

To use the AD-generated gradient in step 2 of Algorithm 2.2 we have to discuss the relation between $J'(u)$ and its counterpart $\mathtt{J}'(\mathtt{u}) = D_{\mathtt{u}}\mathtt{J}(\mathtt{u})$. Note that in $\mathbb{R}^n$ we do not have to distinguish between the derivative or gradient and its Riesz representative. We recall that $\mathtt{J} = J_h \circ \mathcal{I}_h$ and obtain

$$\begin{aligned} \mathtt{J}'(\mathtt{u})\mathtt{e}_i &= D_{\mathtt{u}}((J_h \circ \mathcal{I}_h)\mathtt{u})\mathtt{e}_i &&= D_u J_h(\mathcal{I}_h\mathtt{u}) D_{\mathtt{u}}(\mathcal{I}_h\mathtt{u})\mathtt{e}_i \\ &&&= D_u J_h(u_h)\mathcal{I}_h\mathtt{e}_i = D_u J_h(u_h)\phi_i = (J_h'(u_h), \phi_i), \end{aligned}$$

where $\mathtt{e}_i$ denotes the i-th basis vector in $\mathbb{R}^n$. On the other hand we have with $\mathtt{g} = \mathcal{I}_h^{-1} J_h'(u_h)$:

$$(J_h'(u_h), \phi_i) = \mathtt{g}^T \mathtt{M}\mathtt{e}_i, \quad i = 1, \ldots, n,$$

and therefore

$$\mathtt{J}'(\mathtt{u}) = \mathtt{M}\mathtt{g}. \tag{4.2}$$

Thus the AD-generated discrete function $\mathtt{u} \mapsto \mathtt{J}'(\mathtt{u})$ is used in Algorithm 2.2 in step 2 as follows:

2. (a) Given $\mathtt{u}_0^j$, compute $\bar{\mathtt{g}} = \mathtt{J}'(\mathtt{u}_0^j)$,
   (b) Set $\mathtt{g} = \mathtt{M}^{-1}\bar{\mathtt{g}}$.

Step (b) is necessary because of (4.2) and since Algorithm 2.2 uses $\mathtt{g}$ in step 3. The solving of the linear system can be avoided by slightly modifying step 3:

2. Given $\mathtt{u}_0^j$, compute $\bar{\mathtt{g}} = \mathtt{J}'(\mathtt{u}_0^j)$,
3. Solve

$$(\mathtt{M} + \tau\mathtt{A})\mathtt{x}^{j+1} = \mathtt{M}(\mathtt{x}^j + \tau\mathtt{b}^j + \mathtt{u}_0^j) - \rho\bar{\mathtt{g}}.$$

The difference between the coefficient vectors $\mathtt{g}$ and $\bar{\mathtt{g}} = \mathtt{M}\mathtt{g}$ has to be taken into account when comparing numerical results obtained by both types of discrete gradients, see the last section of this paper.

5 Application of AD on the Navier-Stokes Solver

As pointed out at the end of Section 4 the reverse mode of AD is most appropriate in our case: The discrete cost functional $\mathtt{J}$ has a large number of input variables (the number of points where the distributed control acts, i.e. the number of grid points). On the other hand there is only one output variable, namely the value of $\mathtt{J}$ itself. Thus and because the original source code was provided in FORTRAN the choice of AD software tools was limited. We chose the TANGENT AND ADJOINT MODEL COMPILER (TAMC) in version 5.3.2, which can be accessed remotely, see [14]. Another possible choice would be ODYSSEE [12].

TAMC needs some code preparation. The content of FORTRAN include files has to be inserted in the subroutine bodies. This can be done by a script also available on [14]. Moreover for the reverse mode it is recommended to replace all jump statements (such as *goto*) by other features, e.g. FORTRAN 90 *do-while* loops. In our case only one loop had to be rewritten in this way. Without this change TAMC gives an error message ("irreducible control flow graph"), nevertheless the generated derivative code produced the same (correct) results for the derivative.

We also used the AD tool ADIFOR 2.0 [1], which in this version is only capable of forward mode. On the other hand to our experience (see [13]) it is quite robust and requires no code preparation. Thus we used it for a comparison. We are aware of the fact that the forward mode

is not the right choice for the control problem we study. Therefore any time comparisons for the forward mode are excluded from this paper.

The computational graph from the discrete independent variable, the control u, to the dependent variable, the value J(u) of the discrete cost functional, is depicted in the following diagramme:

$$\begin{array}{ccccc} & (1) & & (2) & \\ \mathtt{u} & \mapsto & \mathtt{x} & \mapsto & \mathtt{J(u)} \\ \text{control} & \textit{state} & \text{state} & \textit{cost} & \text{cost} \\ \text{variable} & \textit{equation} & \text{variable} & \textit{functional} & \\ & \textit{solver} & & \textit{evaluation} & \end{array}$$

Step (2) only requires the evaluation of some norms, i.e. matrix-vector multiplications and scalar products, see (2.2). Step (1) consists of the solution of the discrete counterpart of the first equation in (3.1):

$$(\mathtt{M} + \tau\mathtt{A})\mathtt{x} \quad = \quad \mathtt{x}^j + \tau\mathtt{b}^j + \mathtt{u} \tag{5.1}$$

where $\mathtt{x} = \mathtt{x}^{j+1}$ is the unknown state, $\mathtt{u} = \mathtt{u}^{j+1}$ the control, and $\mathtt{x}^j, \mathtt{b}^j$ given data from the last time step. Since A is the discrete Stokes operator equation (5.1) is said to have quasi-Stokes form. Actually it is *not* solved in a divergence-free space but with standard Taylor-Hood finite elements, that means by introduction of a discrete pressure p. Thus (5.1) is solved as

$$\begin{array}{lcl} (\mathtt{M} + \tau\mathtt{S})\mathtt{v} + \mathtt{B}^T\mathtt{p} & = & \mathtt{v}^j + \tau\mathtt{b}^j + \mathtt{u} \\ \mathtt{Bv} & = & \mathtt{0} \end{array} \tag{5.2}$$

where $\mathtt{x} := (\mathtt{v}, \mathtt{p})$ denotes the pair of coefficient vectors for velocity and pressure, and S is the stiffness matrix of the discrete velocity space. The matrix B comes from the discretization of the negative weak divergence operator. System (5.2) is solved by a Uzawa type algorithm: Formally v is computed from the first equation and inserted in the second. This leads to an equation for p:

$$\mathtt{B}(\mathtt{M} + \tau\mathtt{S})^{-1}\mathtt{B}^T\mathtt{p} \quad = \quad \mathtt{B}(\mathtt{M} + \tau\mathtt{S})^{-1}(\mathtt{v}^j + \tau\mathtt{b}^j + \mathtt{u}). \tag{5.3}$$

This linear system with the positive definite matrix $\mathtt{B}(\mathtt{M}+\tau\mathtt{A})^{-1}\mathtt{B}^T$ is solved by a conjugate gradient (*cg*) method. In every step it requires the multiplication with the system matrix, i.e. the solution of another system with the matrix $\mathtt{M} + \tau\mathtt{A}$ which is positive definite as well. Thus the solution of (5.3) requires two nested *cg* iterations, an outer one on p and an inner one on v.

As pointed out in Section 4 for the reverse mode of AD this implies either recomputation or storage of all intermediate active variables. This is unefficient and unneccessary for converging iterations. Moreover both systems are linear and it should be possible to compute the derivative using the original iteration routine. TAMC provides an easy way to avoid the differentiation of linear self-adjoint operators by inserting directives in the source code. For a subroutine cg which implements a self adjoint solution operator as for example $(\mathtt{M} + \tau\mathtt{A})^{-1}$ these directives have the form

```
cadj subroutine cg
cadj subroutine cg input = 1,3,4,5,6
cadj subroutine cg output = 2
cadj subroutine cg adname = cg
cadj subroutine cg active = 2,3
```

and provide information about subroutine name and the position of input, output, and active variables in the parameter list. Moreover the fourth line implies that TAMC avoids differentiating the subroutine but uses the original one to compute the derivative. This method is also useful for routines whose source code is not available, since their code have not to be passed to TAMC.

Directives of this type can also be used for the outer quasi-Stokes routine that solves (5.2) iteratively for both velocity vector u and pressure p. But here some modification had to be made: Since this iteration returns two output variables TAMC reports an error. This makes sense because a linear and self-adjoint operator can only have one output. Thus it was necessary to introduce the variable $\mathtt{x} = (\mathtt{u}, \mathtt{p})$ which was not used in the original code, i.e. formally to treat equation (5.1).

6 Numerical Results

We present different numerical results: Comparisons of accuracy and performance for the pure gradient evaluation and afterwards for a longer, more realistic control run. We compared:

- the gradient obtained via the adjoint equation,
- finite difference gradients for different stepsizes s,
- an AD-generated gradient where the outer (quasi-Stokes) iteration was algorithmically differentiated, but the inner (cg) iterations were declared as self-adjoint by directives similar to those presented in Section 5.
- an AD-generated gradient where the outer (quasi-Stokes) iteration was declared as self-adjoint. Thus only the norms in the cost functional had to be differentiated algorithmically, whereas for the derivative of the quasi-Stokes solver the original routine was used.

Accuracy

The accuracy of both AD gradients and finite difference approximations with different stepsizes s was compared to the gradient obtained via the adjoint equation, see Table 1. We compared the gradients on a grid with 545 velocity and 145 pressure nodes for different accuracies in the range of 10^{-4} to 10^{-10} for the outer quasi-Stokes iterations. The accuracy for the inner conjugate gradient iterations was always chosen to be 10^{-4} times this value.

The differences between the gradient computed via the adjoint equation and the first AD-generated one (where the quasi-Stokes iteration was algorithmically differentiated) is in the range of the accuracy of this iteration. The results for the second AD version where the outer quasi-Stokes iteration was not algorithmically differentiated differ from the gradient obtained by the adjoint equation only in the range of machine precision. This result is independent of the choice of the accuracy of the quasi-Stokes iteration.

The norms in Table 1 are vector-norms in $\mathbb{R}^n$, the difference in the discrete L^2 norm (i.e. weighted with the mass matrix) are in the range of 10^{-2} for a finite element discretization with gridsize $h \approx 10^{-2}$. The finite difference gradients converge to the gradient obtained by the adjoint equation (and the second AD gradient) until the stepsize h becomes too small.

Performance

To compare the computing time for the gradient obtained via the adjoint equation in Algorithm 2.2 steps 2 and 3 were performed as in Section 3. For the two AD gradients the equivalent are the two steps presented in Section 4. Note that both versions of the AD-generated subroutine compute J and J′, whereas the alternative via the adjoint equation directly computes the new control from which the gradient can be obtained easily. To get the value of the cost functional the discrete norms in J have to evaluated, but the effort for this part is neglegible.

Table 2 compares the cputime for the different gradients measured relative to one functional evaluation for different accuracies in the quasi-Stokes iteration. The gradient computed via the adjoint equation needs about 1.9 of the time of the function evaluation. The first AD-generated code (where the outer quasi-Stokes iteration has been algorithmically differentiated) takes about 3.6 times one function evaluation. This result seems reasonable since a lot of recomputations have to be done in the differentiated quasi-Stokes iteration in this case, compare Section 5. The second AD-generated gradient takes only about 1.3 times the function evaluation und thus is faster than the gradient obtained via the adjoint equation. Naturally the computing time decreases when the accuracy in the iteraive solvers is increased, but the relations between the time needed for function evaluation and the different gradients remain nearly unchanged.

gradient computed by	relative error to adj. eq. gradient computed with an accuracy in the quasi-Stokes iteration of 10^{-10}	10^{-8}	10^{-6}	10^{-4}
forward finite differences $s = 10^{-1}$	1.3e-01	1.3e-01	1.3e-01	1.3e-01
$s = 10^{-2}$	1.3e-02	1.3e-02	1.3e-02	1.3e-02
$s = 10^{-3}$	1.3e-03	1.3e-03	1.3e-03	1.3e-03
$s = 10^{-4}$	1.3e-04	1.3e-04	1.3e-04	1.3e-04
$s = 10^{-5}$	1.3e-05	1.3e-05	1.2e-05	1.3e-05
$s = 10^{-6}$	5.3e-06	7.0e-06	6.4e-06	4.9e-06
$s = 10^{-7}$	4.8e-05	5.3e-05	6.2e-05	4.2e-05
AD: TAMC reverse mode (directives for inner *cg* iterations)	1.3e-15	1.6e-13	3.5e-11	3.8e-09
AD: TAMC reverse mode (directives for quasi-Stokes iteration)	1.8e-16	1.7e-16	1.7e-16	1.9e-16

Table 1: Comparison of accuracy of gradient evaluation computed by finite differences, adjoint equation and AD. All computations were performed in real(8) precision with 1090 independent variables. Errors are in Euklidean vector norm, those for the maximum norm show the same behaviour. The stopping criterion for the inner *cg* iterations is chosen 4 orders of magnitude lower than the one for the quasi-Stokes iteration.

	cputime absolute (and rel. to function eval.) computed with an accuracy in the quasi-Stokes iteration of 10^{-10}	10^{-8}	10^{-6}	10^{-4}
function evaluation	6.5 (1)	5.8 (1)	5.2 (1)	4.4 (1)
gradient computed by - adjoint equation	12.3 (1.89)	11.0 (1.88)	9.6 (1.87)	8.1 (1.83)
- AD: TAMC reverse mode (directives for inner *cg*)	23.0 (3.52)	20.3 (3.50)	17.3 (3.37)	14.4 (3.24)
- AD: TAMC reverse mode (directives for quasi-Stokes)	8.1 (1.23)	7.3 (1.26)	6.7 (1.30)	6.0 (1.34)

Table 2: Comparison of computing time of gradient evaluation computed by finite differences, adjoint equation and AD. All computations were performed on IBM RS/6000 with -O2 optimization in real(8) precision on with 1090 independent variables. Time measurements are mean values over 1000 evaluations. The stopping criterion for the inner *cg* iterations is chosen 4 orders of magnitude lower than the one for the quasi-Stokes iteration.

Complete Control Run

Finally we compared adjoint and AD-generated gradients in a longer, more realistic instantaneous control run on a finer grid with 1024 triangles, 2113 velocity, and 545 pressure nodes. In Alg. 2.1 we use the functional

$$J(u) = \hat{J}(y,u) := \frac{1}{2}\int_\Omega |y-z|^2\,dxdt + \frac{\gamma}{2}\int_\Omega |u|^2\,dxdt.$$

The initial value of the uncontrolled flow is chosen as

$$y(x,0) \;=\; e\begin{bmatrix} (\cos 2\pi x_1 - 1)\sin 2\pi x_2 \\ -(\cos 2\pi x_2 - 1)\sin 2\pi x_1 \end{bmatrix}$$

with e denoting the Euler number, and the desired state z is time dependent and given by

$$z(t,x) = \begin{bmatrix} \varphi_{x_2}(t,x_1,x_2) \\ -\varphi_{x_1}(t,x_1,x_2) \end{bmatrix},$$

where φ is defined through the stream function

$$\varphi(t,x_1,x_2) = \theta(t,x_1)\theta(t,x_2)$$

with

$$\theta(t,y) = (1-y)^2(1-\cos 2k\pi t), \quad y \in [0,1].$$

For the results presented $\gamma = 1.e-2$, $k = 1$ and the time interval is chosen as $[0,2]$, i.e. $T = 2$.

Since the differences between the gradients are very small in one time step, no big discrepancies are to be expected in a longer run. Plots of velocity field of the flow and the corresponding control for different times can be found in Figure 1. For both approaches *first optimize, then discretize* (or: gradient via adjoint equation) on one hand and *first discretize, then optimize* (or: AD gradient) on the other hand differences in this plots are visually indistinguishable.

The relative differences in the cost functional in the control run, using one time the adjoint equation gradient and the other time the AD gradient, are shown in Figure 2.

Values of the cost functional are presented in Figure 3. Note that the velocity z to be tracked has a highly transient behaviour. This explains why the instantaneous control method with starting value $u_0^j = 0$ which we used here for the control has problems to reduce the functional for $t > 0.5$. This can be improved by performing a higher number of gradient steps in every time step (as depicted in Figure 3). An alternative approach is to use an improved starting value for u_0^j, as pointed out in [9, Remark 5.4.2].

Note that the second AD gradient version (that was faster in computing just one gradient evaluation, see Table 2) loses its better performance in the complete control run. The time relation *gradient via adjoint equation : gradient via AD* was now approximately 1 : 1.1. This is due to the additional computational effort in the solution of step 2.(b), i.e. $\mathtt{g} = \mathtt{M}^{-1}\bar{\mathtt{g}}$, compare Section 4. As pointed out there this effort can be avoided by changing step 3.

Acknowledgements

The authors would like to thank Ralf Giering and Thomas Kaminski from FastOpt Hamburg, Germany, for their support in using TAMC.

References

[1] ADIFOR homepage, Argonne National Laboratory, Argonne IL, USA, [http://www-unix.mcs.anl.gov/autodiff/ADIFOR]

[2] Homepage of the AutoDiff project, Argonne Nat. Lab., Argonne IL, USA, [http://www-unix.mcs.anl.gov/autodiff/AD_Tools/index.html]

[3] T. Bewley, H. Choi, R. Temam, and P. Moin (1993). Optimal feedback control of turbulent channel flow. *Center for Turbulence Research Annual Research Briefs.*

[4] H. Choi, M. Hinze, and K. Kunisch (1999). Instantaneous control of backward-facing-step flows. *Applied Numerical Mathematics*, **31**, 133–158.

[5] P. Constantin and C. Foias (1988)- *Navier-Stokes Equations.* The University of Chicago Press.

[6] C.E. García, D.M. Prett, and M. Morari (1989). Model predictive control: Theory and practice - a survey. *Automatica*, **25**(3), 335–348.

[7] R. Giering, and T. Kaminski (1998). Recipes for Adjoint Code Construction. *ACM Transactions on Math. Software*, **24**(4), 437–474.

[8] A. Griewank (2000). *Evaluating Derivatives*, SIAM Frontiers in Appl. Math., Philadelphia.

[9] M. Hinze (1999). *Optimal and instantaneous control of the instationary Navier-Stokes equations.* Habilitationsschrift, Fachbereich Mathematik, Technische Universität Berlin, [http://www.math.tu-dresden.de/~hinze/publications.html]

[10] M. Hinze and S. Volkwein (2002). Instantaneous control for the Burgers equation: Convergence anlysis and numerical implementation. *Nonlinear Analysis, T.M.A.*, **50** (1), 1-26.

[11] S. Kang and H. Choi (2000). Suboptimal feedback control of turbulent flow over a backward-facing step. Submitted to *J. Fluid Mech.*

[12] ODYSSEE homepage. INRIA, France. [http://www-sop.inria.fr/safir/SAM/Odyssee].

[13] T. Slawig (2001). Domain Optimization of a multi-element airfoil using automatic differentiation. *Advances in Engineering Software*, **32**, 225–237.

[14] TAMC homepage. MIT, Boston MS, USA. [http://puddle.mit.edu/~ralf/tamc].

[15] J.B. Rawlings and K.R. Muske (1993). The stability of constrained receding horizon control. *IEEE Transactions on Automatic Control*, **38**(10), 1512–1516.

[16] F. Tröltzsch and A. Unger (2001). Fast solution of optimal control problems in selective cooling of steel. *ZAMM*, **81**, 447–456.

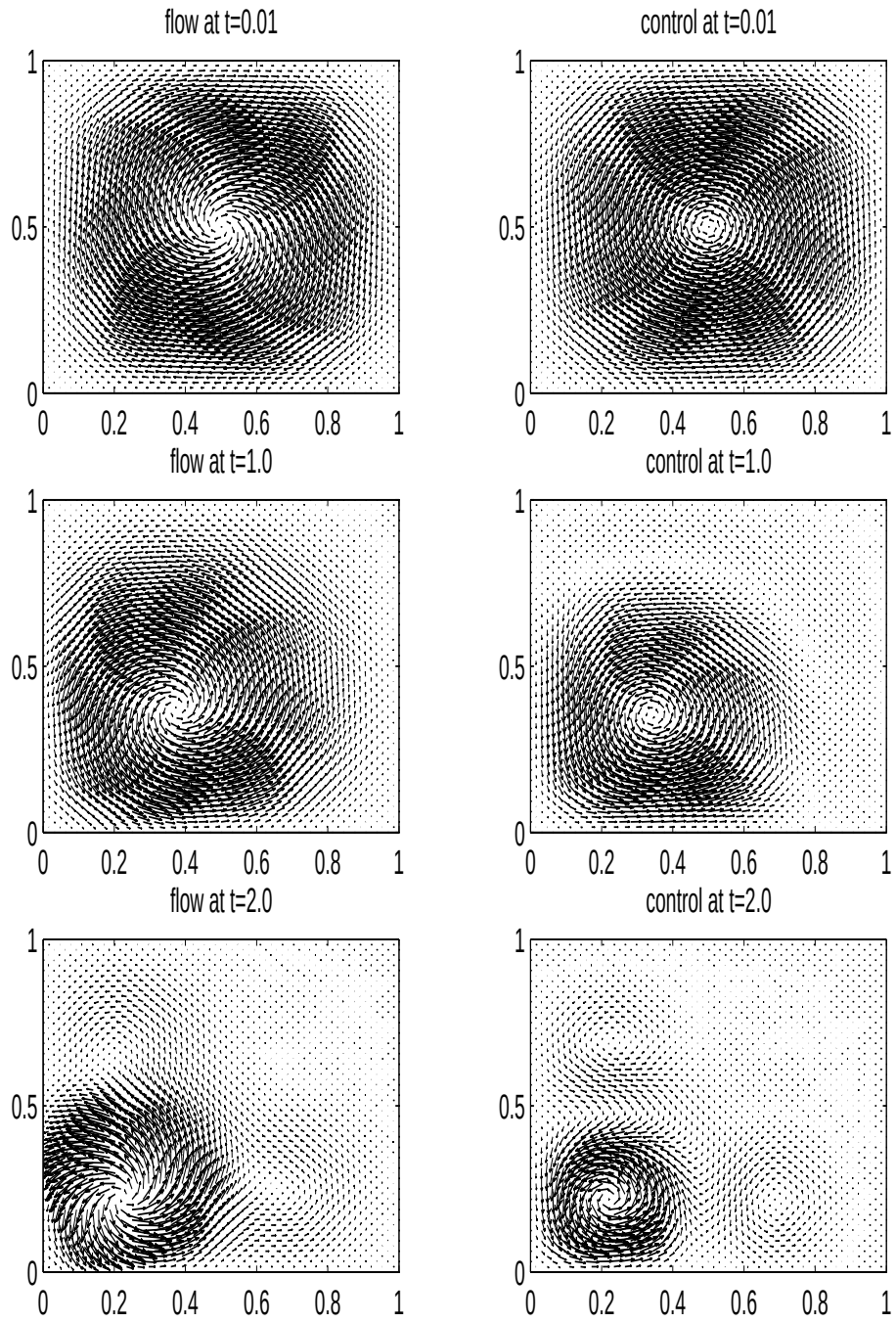

Figure 1: Flow (left) and control (right) at different times in an instantaneous control run. Depicted is the version with the gradient obtained by the adjoint equations, but the two AD versions show no differences.

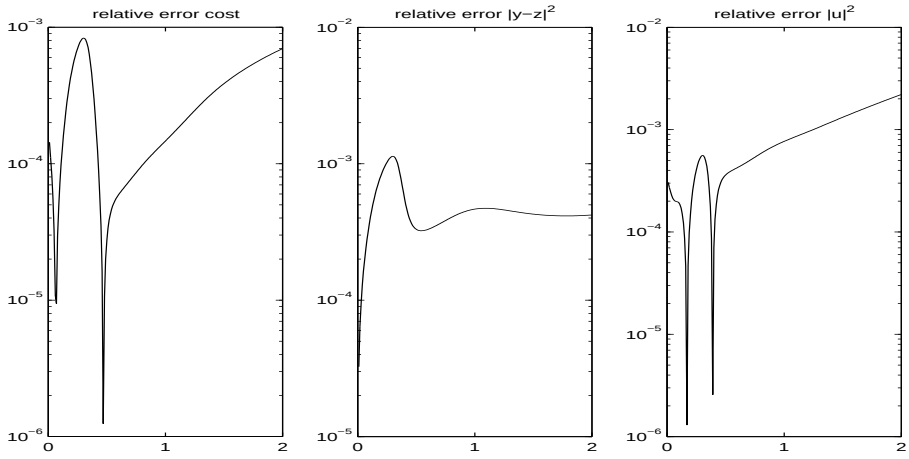

Figure 2: Relative errors of cost functional (left), tracking (middle) and control cost (right) for a control run on $[0, 2]$.

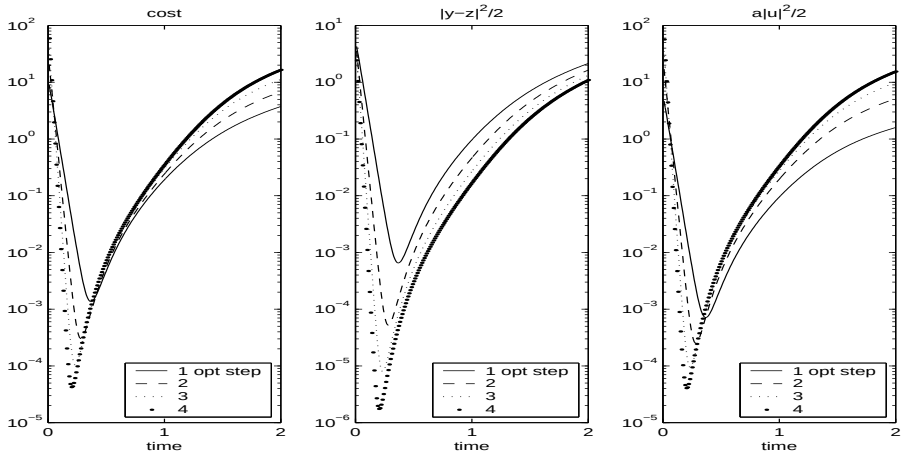

Figure 3: Cost functional (left), tracking (middle) and control cost (right) for a control run on $[0, 2]$ for different numbers of gradient steps in each time step.