Technische Universität Berlin

Institut für Mathematik

PDE-constrained control using COMSOL MULTIPHYSICS – Control of the Navier-Stokes equations

Thomas Slawig[a]

[a]TU Berlin, Institut f. Mathematik, Strasse des 17. Juni 136 MA 4-5, 10623 Berlin, Germany, email: slawig@math.tu-berlin.de, phone: ++49 30 314 28035, fax: ++49 30 314 79706.

Technical Report 2005/26

Preprint-Reihe des Instituts für Mathematik
Technische Universität Berlin

PDE-constrained control using Comsol Multiphysics – Control of the Navier-Stokes equations

Thomas Slawig*

August 8, 2006

Abstract

We show how the software Comsol Multiphysics can be used to solve PDE-constrained optimal control problems. We give a general formulation for such kind of problems and derive the adjoint equation and optimality system. Then these preliminaries are specified for the stationary Navier-Stokes equations with distributed and boundary control. The main steps to define and solve a PDE with Comsol Multiphysics are described. We describe how the adjoint system can be implemented, and how the optimality system can be used by Comsol Multiphysics's built-in functions. Special crucial topics concerning efficiency are discussed. Examples with distributed and boundary control for different type of cost functionals in 2 and 3 space dimensions are presented.

Updated version for Comsol Multiphysics 3.2

The original publication is available at `www.springerlink.com`
Key words: Optimal control, finite element method, Navier-Stokes equations
AMS subject classification: 49K20, 65K10, 65N30, 35Q30, 76D05

1 Introduction

The effective numerical solution of PDE-constrained control problems requires expertise in several disciplines of applied mathematics: Knowledge in the theory of the underlying PDE (e.g. existence and uniqueness results) has to be combined with the choice and analysis of an appropriate control strategy, specifically when adjoint-based algorithms are used. From the numerical point of view, effective discretization schemes and linear or nonlinear solvers for the state equation have to be combined with a suitable optimization method.

Any of these tasks alone may be rather challenging, even more when real-world applications are considered. In many cases, industrial problems are solved numerically by using commercial or legacy software. Its development has incorporated huge amounts of both work and knowledge, and thus it cannot be discarded when proceeding from the pure simulation to mathematical optimization and control. The alternative of plugging in the state equation solver in an off-the-shelve optimization routine – the simplest way of the so-called "first discretize then optimize" approach – is often not very successful, too.

A remedy would be to take advantage of the available state equation (PDE) solver, and combine it with a solver for the analytically derived adjoint equation. This solver has to be implemented. Even though the adjoint equation is always linear, the original PDE solver can not be used in most cases, specifically if the state equation is nonlinear.

In this paper we show how this strategy can be implemented when using the commercial software Comsol Multiphysics ®[2]. Somehow Comsol Multiphysics is a successor of the

*TU Berlin, Institut f. Mathematik, Strasse des 17. Juni 136 MA 4-5, 10623 Berlin, Germany, email: `slawig@math.tu-berlin.de`, phone: ++49 30 314 28035, fax: ++49 30 314 79706.

PDE toolbox in MATLAB ®[8], and it is also widely used in education and university research. On the other hand it offers many typical features of commercial solvers: A graphical user interface, geometry import functions and grid generators for 1,2, and 3 space dimensions, and post-processing facilities. As MATLAB, also COMSOL MULTIPHYSICS is more and more used in real industrial application, at least according to the web site of the distributors. The chances that its application will increase in the future are quite good, since COMSOL MULTIPHYSICS offers a lot of predefined equations and "application modes" (for example for fluid and structural mechanics), variable Finite Element ansatz spaces, and state-of-the-art numerical algorithms for the solution of linear and non-linear systems.

Its main advantage, and the reason why we use COMSOL MULTIPHYSICS here, is that it also allows a more mathematical *equation-based modeling*, i.e. the opportunity to define a PDE by its coefficients or even in weak form, and the chance to invoke commands it from the MATLAB prompt. In this paper we show how these features can be used to design a complete control framework that solves the optimality system of a PDE-constrained control problem as a coupled equation system by the built-in nonlinear solver. Due to the high-level functions provided by the software, the complete control program or script is comparably short. The possibility to call the functions from the command line also allows the use of optimization routines written in MATLAB. We show the structure of a control script for the cases of distributed and boundary control for the stationary incompressible Navier-Stokes equations. Due to the general applicability of the COMSOL MULTIPHYSICS software it is possible to transfer the presented approach to other state equations as well.

The structure of the paper is as follows: In the next section we continue with a general description of PDE constrained control problems with distributed and boundary control. We describe the form of the optimality system including the derivation of the adjoint equation. In Sec. 6 we specialize these results for the test application that we used, the stationary Navier-Stokes equations (NSE). Then we present COMSOL MULTIPHYSICS's solution procedure for the NSE, and describe the basic data, PDE definition and solver structures, as far as they are important for our purpose. We show how the optimality system can be realized in COMSOL MULTIPHYSICS. Afterwards we present efficient model scripts to solve the control problems. We emphasize the crucial points to obtain a really flexible *and* effective framework that may be extended to other state equations as well. We end the paper by showing numerical results and a short summary.

2 PDE-constrained control problems

In this section we present the form of a general PDE-constrained control problem, including the adjoint equation, the optimality system, and a representation of the gradient of the cost. The general form can then be specialized for the given type of state equation. We emphasize that the following computations are formal, and thus for a mathematically correct setting further theoretical investigations are necessary. For details see e.g. [7].

The general form of a PDE-constrained control problem is the following. Let U, Y be Hilbert spaces. The aim is to minimize a functional

$$J : Y \times U \to \mathbb{R}$$

under the constraint

$$e(y, u) \quad = \quad 0, \quad e : Y \times U \to Z, \tag{1}$$

where $y \in Y$ denotes the state and $u \in U$ the control. Clearly y thus depends on u and we will also sometimes write $y = y(u)$ and define $\hat{J}$ as $\hat{J}(u) := J(y(u), u)$. Additional constraints of the form $y \in Y_{ad} \subset Y, u \in U_{ad} \subset U$ (i.e. state and control constraints) may be given as well. The constraint (1) represents a PDE given on a domain Ω with corresponding boundary conditions. We consider only stationary problems here, otherwise there would be an initial condition, too.

A typical example for the cost J is the tracking type functional

$$J(y,u) \quad = \quad \frac{1}{2}\int_{\Omega_c} |y-y_d|^2 dx + \frac{\alpha}{2}\|u\|_U^2 \tag{2}$$

with regularization parameter $\alpha > 0$. Here the control volume Ω_c is a subset of the domain Ω. The functional measures the distance between the solution $y = y(u)$ of the state equation (1) to a given target or desired state y_d. Moreover it takes into account the control cost by the regularization term. In a similar way one may consider a functional that measures this distance only on the boundary $\partial\Omega$ or a part of it. The state equation (1) consists of the PDE and boundary conditions, for distributed control u appears in the PDE, for boundary control in the boundary conditions.

2.1 Adjoint equations and optimality systems

In this subsection we show the general process to get the optimality system or a representation of the gradient of $\hat{J}$. We introduce the Lagrange functional associated with the constrained problem

$$\min_{u\in U} J(y,u) \quad \text{subject to} \quad (1). \tag{3}$$

It is defined as

$$L : Y \times U \times Z^*, \qquad L(y,u,\lambda) := J(y,u) + \langle \lambda, e(y,u)\rangle_{Z^*,Z}$$

where $\langle\cdot,\cdot\rangle_{Z^*,Z}$ denotes the pairing between Z and its dual Z^*. Since in the case of a PDE-constrained problem e is vector-valued (at least due to boundary conditions) we have $Z = Z_1 \times \ldots \times Z_n$. Thus Z^* is isometrically isomorph to the Cartesian product of the duals Z_i^*, i.e. $Z^* \cong Z_1^* \times \ldots \times Z_n^*$, and the dual pairing is given as

$$\langle \lambda, e(y,u)\rangle_{Z^*,Z} \quad = \quad \sum_{i=1}^{n} \langle \lambda_i, e_i(y,u)\rangle_{Z_i^*,Z_i}.$$

The necessary optimality condition for saddle point of L and a minimum of (3) are then computed by setting the directional derivatives of L with respect to (y,u,λ) equal to zero in all admissible directions. We get

$$\left.\begin{array}{rcrcll} L_y(y,u,\lambda)\bar{y} & = & J_y(y,u)\bar{y} + \langle \lambda, e_y(y,u)\bar{y}\rangle_{Z^*,Z} & = & 0 & \forall \bar{y} \in Y \\ L_u(y,u,\lambda)\bar{u} & = & J_u(y,u)\bar{u} + \langle \lambda, e_u(y,u)\bar{u}\rangle_{Z^*,Z} & = & 0 & \forall \bar{u} \in U \\ L_\lambda(y,u,\lambda)\bar{\lambda} & = & \langle \bar{\lambda}, e(y,u)\rangle_{Z^*,Z} & = & 0 & \forall \bar{\lambda} \in Z \end{array}\right\} \tag{4}$$

where the subscript denotes the corresponding partial derivative. The first equation is called the adjoint equation, the second gives a relation between Lagrange multiplier z and the control u, and the third one is just the state equation (1). The adjoint may also be written as

$$e_y(y,u)^*\lambda \quad = \quad -J_y(y,u) \quad \text{in } Z^*,$$

where A^* denotes the adjoint operator of A. The optimality system (4) may be solved directly in the so-called *one-shot* approach. Another alternative is to use a gradient-based iterative algorithm to solve (3). Then the directional derivative of $\hat{J}$ is needed. By the chain rule it is given as

$$\hat{J}'(u)\bar{u} \quad = \quad J_y(y,u)y'(u)\bar{u} + J_u(y,u)\bar{u}.$$

In a similar way the total derivative of (1) with respect to u is

$$e_y(y,u)y'(u)\bar{u} + e_u(y,u)\bar{u} \quad = \quad 0 \quad \forall \bar{u} \in U.$$

Adding the duality pairing of this expression with the Lagrange multiplier λ gives

$$\begin{array}{rcl} \hat{J}'(u)\bar{u} & = & J_y(y,u)y'(u)\bar{u} + \langle \lambda, e_y(y,u)y'(u)\bar{u}\rangle_{Z^*,Z} \\ & & + J_u(y,u)\bar{u} + \langle \lambda, e_u(y,u)\bar{u}\rangle_{Z^*,Z}. \end{array}$$

The first two terms equal zero due to the adjoint equation (with $\bar{y} = y'(u)\bar{u}$). Thus $\hat{J}'$ can be characterized without explicitly knowing $y'(u)$. After solution of both state and adjoint equation

$$\hat{J}'(u)\bar{u} \quad = \quad J_u(y,u)\bar{u} + \langle \lambda, e_u(y,u)\bar{u} \rangle_{Z^*,Z} \tag{5}$$

can be evaluated and used in an iterative optimization algorithm.

3 Application on the Navier-Stokes equations

Now we specialize the results of the last section for distributed and boundary control for the stationary NSE. The equations describe the flow of an incompressible Newtonian fluid in a domain $\Omega \subset \mathbb{R}^d$ with $d \in \{2,3\}$. Unknowns are velocity vector $v = (v_i)_{i=1}^d$ and pressure p. Control problem for the NSE are frequently studied, since they have a wide range of applications. Here we refer to [6],[1],[4][10],[3]. For the theory and numerics of the equations themselves see [5] and [12]. The stationary NSE read

$$\left.\begin{array}{rcll} -\mathrm{div}\,(\nu \nabla v) + v \cdot \nabla v + \nabla p &=& f + u_D & \text{in } \Omega \\ -\mathrm{div}\, v &=& 0 & \text{in } \Omega \\ v &=& g & \text{on } \Sigma_d \\ v &=& u_B & \text{on } \Sigma_c \\ \nu \partial_\eta v - p\eta &=& 0 & \text{on } \Sigma_n \end{array}\right\} \tag{6}$$

The first vector-valued equation represents the balance of momentum, the second one the balance of mass in the fluid. Here f is a given volume force vector, ν the viscosity of the fluid or in dimensionless form the inverse of the Reynolds number. The functions u_D and u_B are the controls. The gradient of v and the nonlinear convective term are defined as

$$\nabla v := \Big(\frac{\partial v_j}{\partial x_i}\Big)_{i,j=1}^d, \qquad v \cdot \nabla w := \Big(\sum_{j=1}^d v_j \frac{\partial w_i}{\partial x_j}\Big)_{i=1}^d.$$

Here the dot denotes the scalar product where the sum is taken over the two adjacent indices of the components of the operands. The boundary conditions are formulated in a general way with

- a part Σ_d with Dirichlet boundary conditions for the velocity. These may be inhomogeneous which refers to an prescribed in- or outflow, or homogeneous which refers to rigid walls with a no-slip condition.
- a part Σ_c with Dirichlet boundary conditions for the velocity, when this is used as control. This refers to blowing/sucking or applying a tangential force on the fluid. For distributed control we set $\Sigma_c = \emptyset$.
- a part Σ_n with a natural boundary conditions. These are incorporated implicitly in the weak formulation, as we will show later. For the NSE this is a mixed condition for the normal derivative $\partial_\eta := \eta \cdot \nabla$ of the velocity and the pressure. This condition represents a free outflow region and is sometimes also referred to as a "do nothing" condition.

Here η denotes the outer normal vector on the boundary $\partial\Omega = \overline{\Sigma_d \cup \Sigma_c \cup \Sigma_n}$. The weak form of the equations is derived in the standard way. We do not include any homogeneous boundary condition in the definition of the ansatz space which would also be possible, e.g. choose $H_1^0(\Omega)^d$ in the case $\partial\Omega = \Sigma_d$ and $g = 0$. We denote by $V := H^1(\Omega)^d$ the Sobolev space of vector-valued functions with values and first derivatives in $L^2(\Omega)$. As test space we use

$$W \quad := \quad \{w \in V = H^1(\Omega)^d : w|_{\Sigma_d \cup \Sigma_c} = 0\}.$$

The pressure is uniquely defined in $P := L^2(\Omega)$, except for $\Sigma_n = \emptyset$ where this is the case only in

$$P := L_0^2(\Omega) \quad := \quad \{p \in L^2(\Omega) : \int_\Omega p dx = 0\},$$

which is isometrically isomorph to $L^2(\Omega)/\mathbb{R}$. In computations uniqueness of the pressure is usually realized by fixing the value to zero in one node. The velocity is unique only if ν is big enough in relation to the norm of f, compare [5, Theorem IV.2.4]. The outflow condition is the natural boundary condition when applying Green's formula to the Laplace operator for v and the gradient of p, compare [5, I.(2.17),(2.18)]. Then we obtain the weak formulation: Find $(v,p) \in V \times P$ as solution of

$$\begin{aligned} \nu(\nabla v, \nabla w)_\Omega + (v \cdot \nabla v, w)_\Omega - (p, \operatorname{div} w)_\Omega - (f + u_D, w)_\Omega &= 0 \quad \forall w \in W \\ -(\operatorname{div} v, q)_\Omega &= 0 \quad \forall q \in P \\ v - g &= 0 \quad \text{on } \Sigma_d \\ v - u_B &= 0 \quad \text{on } \Sigma_c \end{aligned} \tag{7}$$

Here $(\cdot,\cdot)_\Omega$ is the L^2 inner product, where we do not distinguish in the notation between scalar-, vector-, and tensor- or matrix-valued functions. For vector-valued functions v, w we use $(v,w)_\Omega := \sum_{i=1}^d (v_i, w_i)_\Omega$, for matrix-valued functions $(v,w)_\Omega := \sum_{i,j=1}^d (v_{ij}, w_{ij})_\Omega$.

To fit the equations in the framework of the last section the two remaining boundary conditions are formulated in weak form in $L^2(\Sigma_d)^d$ and $L^2(\Sigma_c)^d$, respectively. Then (7) has the form (1) where e has four components. We thus get a Lagrange multiplier $(\lambda_1, \lambda_2, \lambda_3, \lambda_4) \in Z^* = (W \times P \times L^2(\Sigma_d)^d \times L^2(\Sigma_c)^d)^* \cong W \times P \times L^2(\Sigma_d)^d \times L^2(\Sigma_c)^d$. The fourth component is only needed for boundary control. Note that $P^* \cong P$ in both cases.

3.1 Lagrange functional and adjoint equation

With $y := (v,p) \in V \times P =: Y, u := (u_D, u_B)$ the Lagrangian has the form

$$\begin{aligned} L(y,u,\lambda) &= J(v,p,u) + \nu(\nabla v, \nabla \lambda_1)_\Omega + (v \cdot \nabla v, \lambda_1)_\Omega - (p, \operatorname{div} \lambda_1)_\Omega \\ &\quad -(f + u_D, \lambda_1)_\Omega - (\operatorname{div} v, \lambda_2)_\Omega + (v - g, \lambda_3)_{\Sigma_d} + (v - u_B, \lambda_4)_{\Sigma_c}. \end{aligned}$$

The adjoint system is obtained by computing the derivative in direction $\bar{y} = (w,q)$, compare the first equation in (4), as

$$\begin{aligned} \left.\begin{aligned} \nu(\nabla w, \nabla \lambda_1)_\Omega + (v \cdot \nabla w + w \cdot \nabla v, \lambda_1)_\Omega \\ -(\operatorname{div} w, \lambda_2)_\Omega + (w, \lambda_3)_{\Sigma_d} + (w, \lambda_4)_{\Sigma_c} \end{aligned}\right\} &= -J_v(v,p,u)w \quad \forall w \in V \\ -(\operatorname{div} \lambda_1, q)_\Omega &= -J_p(v,p,u)q \quad \forall q \in P \end{aligned} \tag{8}$$

Using Green's formula we obtain (with $\lambda_1 \in W$ and thus $\lambda_1|_{\Sigma_d \cup \Sigma_c} = 0$):

$$\begin{aligned} \nu(\nabla w, \nabla \lambda_1)_\Omega &= -(\operatorname{div}(\nu \nabla \lambda_1), w)_\Omega + \nu(\partial_\eta \lambda_1, w)_{\Sigma_n \cup \Sigma_d \cup \Sigma_c} \\ -(\operatorname{div} w, \lambda_2)_\Omega &= (\nabla \lambda_2, w)_\Omega - (\lambda_2 \eta, w)_{\Sigma_n \cup \Sigma_d \cup \Sigma_c} \\ (v \cdot \nabla w, \lambda_1)_\Omega &= -(v \cdot \nabla \lambda_1, w)_\Omega + ((v \cdot \eta)\lambda_1, w)_{\Sigma_n} \end{aligned}$$

see [5, I.(2.17-18), Lemma IV.2.2] and [11, Lemma 6.3]. Moreover

$$(w \cdot \nabla v, \lambda_1)_\Omega = \int_\Omega \sum_{i,j} w_j \frac{\partial v_i}{\partial x_j} \lambda_{1i} dx = ((\nabla v) \cdot \lambda_1, w)_\Omega,$$

which corresponds to our convention for the dot product. Some authors use the notation $(\nabla v)^T \lambda_1$ here. Testing the first equation of (8) with $w \in H_0^1(\Omega)^d$ all boundary terms vanish and we get the equation

$$-\operatorname{div}(\nu \nabla \lambda_1) - v \cdot \nabla \lambda_1 + (\nabla v) \cdot \lambda_1 + \nabla \lambda_2 = -J_v(v,p,u) \quad \text{in } \Omega.$$

Using this and testing (8) with $w \in V, w|_{\Sigma_d \cup \Sigma_c} = 0$ gives

$$\nu \partial_\eta \lambda_1 - \lambda_2 \eta + (v \cdot \eta)\lambda_1 = 0 \quad \text{on } \Sigma_n,$$

which is the condition for the Lagrange multipliers λ_1, λ_2 on the outflow boundary. Choosing $w \in V, w|_{\Sigma_n \cup \Sigma_d} = 0$ we get the representation

$$\lambda_4 \quad = \quad -(\nu \partial_\eta \lambda_1 - \lambda_2 \eta) \quad \text{on } \Sigma_c$$

which is used in the boundary control case. A similar relation holds for λ_3, but this multiplier has no relevance here. Re-formulated as PDE and with $\lambda := \lambda_1, \mu := \lambda_2$ the adjoint system reads

$$\begin{array}{rcll} -\mathrm{div}\,(\nu \nabla \lambda) - v \cdot \nabla \lambda + (\nabla v) \cdot \lambda + \nabla \mu & = & -J_v(v,p,u) & \text{in } \Omega \\ -\mathrm{div}\,\lambda & = & -J_p(v,p,u) & \text{in } \Omega \\ \lambda & = & 0 & \text{on } \Sigma_d \cup \Sigma_c \\ \nu \partial_\eta \lambda - \mu \eta + (v \cdot \eta)\lambda & = & 0 & \text{on } \Sigma_n. \end{array} \tag{9}$$

For distributed control we set $u_B = 0, \Sigma_c = \emptyset$, and $U = L^2(\Omega)^d$ as control space. The second equation in (4) gives

$$J_u(y, u_D) - \lambda \quad = \quad 0 \quad \text{in } \Omega$$

and thus

$$u_D \quad = \quad \lambda/\alpha \tag{10}$$

if the regularization term is chosen as in (2). Inserting this in the first equation of (7) this system together with (8) forms a coupled optimality system that can be solved in the one shot approach. For an iterative approach the derivative of $\hat{J}$ is computed from (5) as

$$\hat{J}'(u_D) \quad = \quad J_u(y, u_D) - \lambda. \tag{11}$$

For boundary control we set $u_B = 0$, and as control space $U = H_{00}^{1/2}(\Sigma_c)^d$. The second equation in (4) gives

$$J_u(y, u_B) - \lambda_4 \quad = \quad 0 \quad \text{on } \Sigma_c$$

and thus

$$u_B \quad = \quad -(\nu \partial_\eta \lambda - \mu \eta)/\alpha \quad \text{on } \Sigma_c \tag{12}$$

if the regularization term is chosen as in (2). In an iterative scheme one may use

$$\hat{J}'(u_B) \quad = \quad J_u(y, u_B) - \lambda_4. \tag{13}$$

4 Solving the Navier-Stokes equations in COMSOL MULTIPHYSICS

COMSOL MULTIPHYSICS is a software that solves PDEs using the finite element method in one to three space dimensions, using elements of arbitrary oder (up to four). It has a graphical user interface and several built-in application modes. With them the user can solve a lot of application problems without defining the PDE itself, just by defining the geometry, adding problem-specific coefficients, choosing the boundary conditions and so on. These are features that most industrial codes offer. What is usually not predefined are the adjoint equations that are necessary when doing PDE-constrained control, and not using finite difference approximations or algorithmic differentiation to compute the derivatives of the cost functional. But since COMSOL MULTIPHYSICS allows the user to define an almost arbitrary PDE, all its solution, post-processing features can be used also to solve them. The modeling of the adjoint equation is possible using the graphical interface as well. In addition COMSOL MULTIPHYSICS routines can be called from the MATLAB command line (after typing `comsol matlab` in a shell) and thus can be coupled with all MATLAB

```
 1  fem.geom = rect2(0,1,0,1);
 2  fem.mesh = meshinit(fem,'hauto',5);
 3  fem.dim = {'v1' 'v2' 'p'};
 4  fem.shape = [2 2 1];
 5  fem.pnt.constr = {{0 0 '-p'} 0};
 6  fem.pnt.shape = [1,2,3];
 7  fem.pnt.ind = [1 2 2 2];
 8  fem.form = 'general';
 9  fem.const.nu = 1;
10  fem.equ.expr = {'F1' '0' 'F2' '0'};
11  fem.equ.ga={{{'p-nu*v1x' '-nu*v1y'}{'-nu*v2x' 'p-nu*v2y'}{0 0}}};
12  fem.equ.f = {{'F1-v1*v1x-v2*v1y' 'F2-v1*v2x+v2*v2y' 'v1x+v2y'}};
13  fem.bnd.ind = [2 2 1 2];
14  fem.bnd.expr = {'g1' '1.0'};
15  fem.bnd.r = { {'v1-g1' 'v2'} {'v1' 'v2'} };
16  fem = femdiff(fem);
17  fem.xmesh = meshextend(fem);
18  fem.sol = femnlin(fem,'Ntol',1e-4);
19  postflow(fem,{'v1' 'v2'})
```

Figure 1: COMSOL MULTIPHYSICS script for the solution of a driven cavity flow in 2-D. Line numbers are not part of the code.

routines, for example optimization functions. This special feature shall be exploited, and we thus will describe here how state and adjoint equations can be defined and solved using the commands on the MATLAB prompt.

In this section we describe at first how the NSE themselves are solved. This will show the basic data and function structure of COMSOL MULTIPHYSICS. All relevant information is contained in one MATLAB structure which we call `fem` here, analogously to the notation in the COMSOL MULTIPHYSICS manual. Its name is arbitrary, whereas the names of the substructures are fixed. The whole process of solving a PDE can be split up into the following steps, which can be seen in Fig. 1 for our first test example, the driven cavity flow:

4.1 Geometry and mesh

Definition (or import) of the geometry is the first step, here we defined a square $[0,1]^2$ (line 1). The mesh initialization follows (line 2). The optional parameter `hauto` controls the mesh-size (from `1`: fine to `9`: coarse). Further refinement is possible using the command `fem.mesh = meshrefine(fem)`.

4.2 Definition of the equation variables and their finite element type

The ansatz function type and order have to be specified (lines 3-7):

- We want the unknowns to be named as `v1`, `v2`, `p` instead of the default names `u1`, `u2`, `u3` (line 3, optional).
- To satisfy the inf-sup or LBB condition, a stability condition that ensures solvability of a discretized NSE scheme (see [5, Theorem I.1.1]) we choose second order elements for the velocity and first order for the pressure (line 4). This element pair is known under the name Taylor-Hood. Elements up to order four can be chosen here.
- Since for the cavity flow we set $\Sigma_n = \emptyset$ we need to use $L^2(\Omega)/\mathbb{R}$ as pressure space. Thus one pressure value is fixed to zero, and this point constraint is set in lines 5-7.

4.3 Definition of the PDE

At first the form of the equation has to be set (line 9). COMSOL MULTIPHYSICS has three options:

- `coefficient`, the default, which is appropriate only for linear problems and corresponds to a classical formulation of the PDE,
- `general`, which uses the classical formulation in divergence form and is appropriate for nonlinear equations,
- `weak`, which allows (and requires) the user to write the equation in the weak form, i.e. using the test functions w, q from (7).

It is also possible to add weak terms to an equation defined in general form, a very convenient feature for example when adding upwind stabilization, a technique that is required in particular for small values of ν. We use the `general` form which is suitable for both the NSE and the adjoint equation. The PDE coefficients have to be defined in a substructure named `equ` of the main structure `fem`.

In general form a PDE system with n equations for the vector of unknowns $(u_1, \ldots, u_N)$ in COMSOL MULTIPHYSICS is written as

$$\operatorname{div} \Gamma_l \quad = \quad F_l, \quad l = 1, \ldots, N.$$

We present the settings for the two-dimensional case, the extension to $d = 3$ is straight-forward. For simplicity and in accordance to COMSOL MULTIPHYSICS's notation we set $x = (x, y) \in \mathbb{R}^2$ for the spatial coordinates. The unknowns are $(u_1, u_2, u_3) := (v_1, v_2, p)$. A subscript x or y denotes the partial derivative of an unknown with respect to x, y, respectively. COMSOL MULTIPHYSICS uses the notation `v1x` for v_{1x} etc. The vector-valued momentum equation and the continuity equation of (6) together give $N = 3$ scalar equations:
$l = 1$:

$$\operatorname{div}\left(-\nu\nabla v_1 + \begin{bmatrix} p \\ 0 \end{bmatrix}\right) = \frac{\partial}{\partial x}(-\nu v_{1x} + p) + \frac{\partial}{\partial y}(-\nu v_{1y}) = f_1 - (v_1 v_{1x} + v_2 v_{1y})$$

$l = 2$:

$$\operatorname{div}\left(-\nu\nabla v_2 + \begin{bmatrix} 0 \\ p \end{bmatrix}\right) = \frac{\partial}{\partial x}(-\nu v_{2x}) + \frac{\partial}{\partial y}(-\nu v_{2y} + p) = f_2 - (v_1 v_{2x} + v_2 v_{2y})$$

$l = 3$:

$$0 \quad = \quad v_{1x} + v_{2y}.$$

Thus we obtain

$$\Gamma \quad = \quad (\Gamma_l)_{l=1}^3 = \left\{ \begin{bmatrix} -\nu v_{1x} + p \\ -\nu v_{1y} \end{bmatrix}, \begin{bmatrix} -\nu v_{2x} \\ -\nu v_{2y} + p \end{bmatrix}, \begin{bmatrix} 0 \\ 0 \end{bmatrix} \right\}, \tag{14}$$

$$F \quad = \quad (F_l)_{l=1}^3 = \begin{bmatrix} f_1 - (v_1 v_{1x} + v_2 v_{1y}) \\ f_2 - (v_1 v_x + v_2 v_{2y}) \\ v_{1x} + v_{2y} \end{bmatrix} \tag{15}$$

Note that $\nabla p = \operatorname{div}(pI)$ where I is the $(d \times d)$ identity tensor or matrix. The definition of Γ and F is not unique, it would also be possible to put the coefficients of the third equation in Γ_3 and not in F_3. COMSOL MULTIPHYSICS uses a weak formulation and performs an integration by parts on the terms in Γ. Thus the above choice corresponds to the weak formulation (7).

Both Γ and F have to be defined in the substructures `ga` and `f`, respectively, of `fem.equ` as *cell arrays*, indicated by the use of the brackets. The entries are defined as strings. The rows or components Γ_l of Γ have to be concatenated by extra pairs of brackets, see lines 11-12 in Fig. 1.

Note also that for both Γ and F there are two outer pairs of brackets. COMSOL MULTIPHYSICS allows the user to define different regions in the computational domain with different PDE settings in each region (so-called *multi-physics*). For each of these regions different coefficients Γ and F may be defined, and they are concatenated in by the outmost pairs of brackets. Although we do not use this feature here, still this additional pair of brackets is required.

The constant ν that is used in Γ has to be defined as element of the substructure `fem.const` before (see line 9).

In line 10 we defined, again as cell array of strings, the right-hand side vector $f = (f_1, f_2)$ to be zero. For this purpose the expressions `F1`, `F2` have been defined in the substructure `expr` of the structure `fem.equ`, and they have been given the value `'0'` as string. It is also possible to insert the inhomogeneities directly in the array F, without using the substructure `expr`. Using the spatial coordinates x, y and any of MATLAB's built-in functions arbitrary inhomogeneities can be defined. The definition of user-defined functions will be described later on.

4.4 Boundary conditions

The substructure `fem.bnd` defines the boundary conditions. Based on the initial geometry different boundary sections can be defined (line 13): The cavity initially has four edges, on one of them (edge 3, the upper one) we impose inhomogeneous Dirichlet conditions, on the other three we have homogeneous Dirichlet conditions. The numbering of the edges has to be found by trial and error, since it has no obvious order. The indices of the boundary parts of the edges have to be put in the array `fem.bnd.ind`.

The boundary conditions on each boundary section are written as

$$\left.\begin{array}{rcll} -\eta\cdot\Gamma_l &=& G_l + \displaystyle\sum_{m=1}^{M} \frac{\partial R_m}{\partial u_l}\mu_m, & l = 1,\ldots,N \\ R_m &=& 0, & m = 1,\ldots,M \end{array}\right\} \quad \text{on } \partial\Omega. \tag{16}$$

The μ_m are artificially introduced Lagrange multipliers, i.e. additional free variables. Depending on the choice of the vectors $R = (R_m)_{m=1}^M$ and $G = (G_l)_{l=1}^N$ (implemented in `fem.bnd.r` and `fem.bnd.g`, both zero by default) Dirichlet, natural and mixed boundary conditions can be realized.

To define Dirichlet conditions $G = 0$ is set. Then the first equation in (16) imposes no condition on the unknowns because of the free Lagrange multipliers. By defining R the Dirichlet conditions are specified. For the example of the driven cavity we have

$$\begin{array}{llllll} v_1 = g_1, & v_2 = 0 & \text{i.e.} & R_1 = v_1 - g_1, & R_2 = v_2 & \text{on } \Sigma_{d1} \\ v_1 = 0, & v_2 = 0 & \text{i.e.} & R_1 = v_1, & R_2 = v_2 & \text{on } \Sigma_{d2} \end{array}$$

for some given function g_1. This function (here set to the constant 1.0) and the vector R are implemented in lines 14-15, where `fem.bnd.r` is a cell array with two components, one for every boundary section defined in `fem.bnd.ind`.

For a boundary section with free outflow (or "do nothing") condition $R = 0$ is set. The Lagrange multipliers are meaningless, and since

$$\begin{array}{rcccccccl} -\eta\cdot\Gamma_1 &=& -\eta\cdot\left(-\nu\nabla v_1 + \begin{bmatrix} p \\ 0 \end{bmatrix}\right) &=& \nu\partial_\eta v_1 - p\eta_1 &=& G_1, \\ -\eta\cdot\Gamma_2 &=& -\eta\cdot\left(-\nu\nabla v_2 + \begin{bmatrix} 0 \\ p \end{bmatrix}\right) &=& \nu\partial_\eta v_2 - p\eta_2 &=& G_2, \\ -\eta\cdot\Gamma_3 &=& 0 & & & = & G_3 \end{array} \tag{17}$$

in the NSE case the first equation in (16) gives the desired natural (here free outflow) condition if $G = 0$ is set. An example is given in Fig. 2.

```
 1  fem.geom = rect2(0,3,0,0.5) - rect2(0,0.5,0,0.25);
13  fem.bnd.ind = [1 2 2 2 2 3];
14  fem.bnd.expr = {'g1' '1-64*(y-0.375)*(y-0.375)'};
15  fem.bnd.r = { {'v1-g1' 'v1'} {'v1' 'v2'} {0 0} };
```

Figure 2: Implementation of free outflow boundary conditions for a backward facing step in $\Omega :=]0,3[\times]0,0.5[\backslash[0,0.5]\times[0,0.25]$. The remaining lines are as in Fig. 1, except lines 5-7 which are not needed any more. The three boundary sections defined in line 13 are the inflow (1), the wall (2), and the outflow (3) region. The expression `g1` describes a parabolic inflow as a function of the spatial coordinate y.

4.5 Assembly and solution of the discrete system

The `femdiff` command (line 16) symbolically computes a linearization of the nonlinear terms for the solver. The routine `meshextend` (line 17) computes additional nodes if basis functions of order higher than one are used. Then it assembles the discrete system.

The nonlinear solver (line 18) is a damped Newton method and has a lot of parameters, here we only changed the default of the stopping criterion. As solver for the linear systems in every Newton step the routine uses the sparse LU method UMFPACK, iterative schemes as GMRES are available. Note that the linearized NSE system is indefinite and non-symmetric, such that many Krylov methods are not applicable. The LU decomposition is the bottleneck of the method with respect to storage requirements.

We show just a streamline plot (line 19, see Fig. 5) of the solution as example for the post processing, more options are available. For more details on the used commands and further options we refer to the COMSOL MULTIPHYSICS documentation.

5 Implementation of the adjoint equation

The adjoint system (9) can be implemented in a rather similar way as the NSE themselves. Since it is a linear equation it would be possible to use the `coefficient` from to define the PDE. But this has no advantage, the linear solver can treat the `general` form as well.

Using the variables $(u_1,u_2,u_3) := (\lambda_1,\lambda_2,\mu)$ with $\lambda = (\lambda_1,\lambda_2)$ the coefficient Γ has the same form as in (14), just with (v_1,v_2,p) replaced by $(\lambda_1,\lambda_2,\mu)$. The form of F is derived using

$$\begin{aligned} v\cdot\nabla\lambda &= \begin{bmatrix} v_1 \\ v_2 \end{bmatrix} \cdot \begin{bmatrix} \lambda_{1x} & \lambda_{2x} \\ \lambda_{1y} & \lambda_{2y} \end{bmatrix} = \begin{bmatrix} v_1\lambda_{1x}+v_2\lambda_{1y} \\ v_1\lambda_{2x}+v_2\lambda_{2y} \end{bmatrix}, \\ \nabla v\cdot\lambda &= \begin{bmatrix} v_{1x} & v_{2x} \\ v_{1y} & v_{2y} \end{bmatrix} \cdot \begin{bmatrix} \lambda_1 \\ \lambda_2 \end{bmatrix} = \begin{bmatrix} v_{1x}\lambda_1+v_{2x}\lambda_2 \\ v_{1y}\lambda_1+v_{2y}\lambda_2 \end{bmatrix}. \end{aligned}$$

Thus we get

$$F = \begin{bmatrix} J_1 + v_1\lambda_{1x} + v_2\lambda_{1y} - (v_{1x}\lambda_1 + v_{2x}\lambda_2) \\ J_2 + v_1\lambda_{2x} + v_2\lambda_{2y} - (v_{1y}\lambda_1 + v_{2y}\lambda_2) \\ J_3 + \lambda_{1x} + \lambda_{2y} \end{bmatrix}, \tag{18}$$

where $(J_1,J_2) = -J_v(v,p,u)$ and $J_3 = -J_p(v,p,u)$.

The boundary conditions are simple in the case of Dirichlet conditions, in the case of the free outflow or natural condition the last equation in (9) can be implemented using the vector G in (17), compare Fig. 3.

The main challenge is that we need the solution (v,p), i.e. the COMSOL MULTIPHYSICS variables `v1`, `v2`, `p` from the state equations as coefficients to compute the adjoint variables (λ,μ). It is possible to store the values of `v1`, `v2`, `p` after the solution of the state equations in MATLAB variables, but it is not possible to define such a variable as COMSOL MULTIPHYSICS expression and use it as coefficient.

```
15  fem.bnd.r = { {'la1' 'la2'} {'la1' 'la2'} {0 0} };
    fem.bnd.g = { {0 0 0} {0 0 0} {'-v1*la1' -'v1*la2' 0} };
```

Figure 3: Implementation of the modified free outflow condition from (9) for the adjoint variables $(\lambda_1, \lambda_2, \mu) \hat{=} (\texttt{la1}, \texttt{la2}, \texttt{mu})$. Here for the backward facing step flow (compare Fig. 2) the outer normal vector on the outflow boundary is $\eta = (1, 0)$. Thus $(v \cdot \eta)\lambda = v_1 \lambda$ in the last equation of (9).

```
20 global v1g
21 v1g = posteval(fem,'v1');
22 fcns{1}.type='inline'; fcns{1}.name='v1(x,y)';
23 fem.functions = fcns;
24 fem.equ.expr = {'J1' '-v1(x,y)' 'J2' '-v2(x,y)' 'J3' '0'};
25 fem.equ.ga = { { {'mu-nu*la1x'   '-nu*la1y'} ...
                    {'  -nu*la2x' 'mu-nu*la2y'} {0 0} } };
26 fem.equ.f={{ 'J1+v1(x,y)*la1x+v2(x,y)*la1y-v1x(x,y)*la1-v2x(x,y)*la2'...
                'J2+v1(x,y)*la1x+v2(x,y)*la2y-v1y(x,y)*la1-v2y(x,y)*la2'...
                'J3+la1x+la2y'} };
function f = v1(x,y)
  global v1g
  f = griddata(v1g.p(1,:),v1g.p(2,:),v1g.d,x,y);
```

Figure 4: Storing the value of v_1, i.e. `v1` in a global variable `v1g` and using it in a COMSOL MULTIPHYSICS function (line 22) for the derivative of the functional $J = \frac{1}{2}\|v\|^2_{L^2(\Omega)^2}$. The function `griddata` interpolates the data given on the COMSOL MULTIPHYSICS mesh on the mesh point (x, y) itself. This is responsible for the high computational effort when assembling the discrete systems. The variable `v1g` is a struct that contains grid points and values. We show only one function here, the other coefficients in (18), compare line 26, have also be defined.

One remedy is to make the variable global, define a function that evaluates it, and declare and use this as a COMSOL MULTIPHYSICS function. This has to be done for all needed coefficients in Γ and F for the adjoint system, thus also for the partial derivatives of v. This basic procedure is depicted in Fig. 4. The problem is the immense temporal effort when assembling the system matrices, since the function has to be evaluated and thus the state variable to be interpolated on every grid point.

The situation gets even more complicated when a tracking type functional (2) is used. It needs one or more given functions $y_d = (v_{1d}, v_{2d})$, for example results from previous computations with lower Reynolds number. Also these quantities have to be realized as COMSOL MULTIPHYSICS functions. The effort to assemble the inhomogeneities thus grows even more.

The much more efficient and elegant solution is to build up a system consisting of both the state and the adjoint equations. This system can then be solved in two different ways:

- as a fully coupled system, inserting the values for the controls as inhomogeneity and/or boundary terms in the state equations and then solving for state and adjoints together (one shot approach),
- sequentially by solving first only for the state variables and afterwards for the adjoints, updating the controls and so forth (iterative approach).

Both approaches will be discussed in the next section.

6 Control using COMSOL MULTIPHYSICS

In this section we show how a control problem for the stationary NSE can be solved with COMSOL MULTIPHYSICS. The two approaches mentioned above are presented. We show that the one shot approach is quite effective, whereas an iterative approach does not fit very well to COMSOL

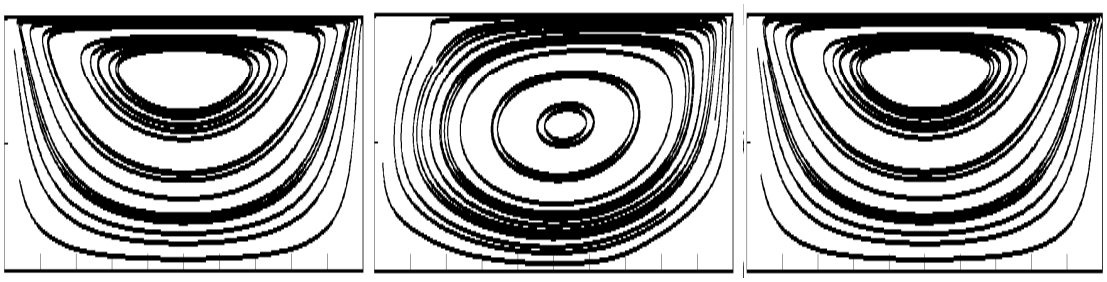

Figure 5: Uncontrolled driven cavity flow for $\nu = 1$ (desired state, left), uncontrolled (middle) and controlled for $\nu = \frac{1}{1000}$. The cost (without regularization term) was reduced to 0.5 % compared to the uncontrolled flow. 1625 pressure nodes.

MULTIPHYSICS's built-in PDE representation and data structures. Examples for distributed and boundary control implementations are given.

6.1 One shot approach

The optimality system (4) summarizes the necessary conditions for a solution of the constrained problem (3). Thus one may try to solve it directly. Then the second equation which gives the relation between adjoint variable and control is inserted for the control in the state equation, and a coupled system of state and adjoint equation is solved.

Following the idea to solve the whole optimality system and using the coupling information between adjoints and control from (10) leads to rather short implementations that we present in the following subsections.

If the state equation is non-linear an appropriate solver is Newton's method (note that the adjoint equation is always linear). The resulting optimization algorithm is a variant of the SQP method, see for example [9]. In every step of the newton iteration a linear system has to be solved. The one shot approach doubles the size of the whole system, and thus also of the linearized systems to be solved in each Newton step. For an iterative solver storing issues may become crucial here.

6.1.1 Distributed control for driven cavity flow.

The first example that we consider is a typical test case for the NSE. The computational domain is the unit square, where on the upper and sometimes also on the lower part of the boundary a velocity field is prescribed. The two lateral boundaries are considered as wall, thus there homogeneous Dirichlet boundary conditions are imposed. There is no outflow boundary, i.e. $\Sigma_n = \emptyset$. If a constant positive horizontal velocity is imposed on the top and v is set to zero at the bottom, the flow shows one big eddy turning clockwise. Depending on the Reynolds number the center of this vortex moves to the right and its shape slightly changes.

The COMSOL MULTIPHYSICS script is shown in Fig. 6 for distributed control, compare also Fig. 1. Setting $\nu = \frac{1}{1000}$ we tried to achieve the flow for $\nu = 1$ by imposing distributed control on the whole domain. Since we use a tracking type functional we add a third set of equations for the desired state and solve for it firstly. The results are shown in Fig. 5 and 9.

In this example the one shot approach with COMSOL MULTIPHYSICS's damped Newton algorithm worked very well, it converged in six steps for a regularization parameter of 0.01, and produced a satisfying result, even without choosing a special initial value for the nonlinear iteration.

6.1.2 Boundary control for backward facing step channel flow.

In this second example we try to minimize the region of back-flow in a channel with a backward facing step. Free outflow conditions were imposed at the end of the channel. For low values of ν

```
 3 fem.dim = {'v1d' 'v2d' 'pd' 'v1' 'v2' 'p' 'la1' 'la2' 'mu'};
 4 fem.shape = [2 2 1 2 2 1 2 2 1];
 5 fem.pnt.constr = { {0 0 '-pd' 0 0 '-p' 0 0 '-mu'} 0};
10 fem.equ.expr = {'J1' 'v1d-u' 'J2' 'v2d-v'};
11 fem.equ.ga = { { {'pd-nu*v1dx'   '-nu*v1dy'} ...
                    {  '-nu*v2dx' 'pd-nu*v2dy'} {0 0} ...
                    {'p-nu*v1x'  '-nu*v1y'} ...
                    { '-nu*v2x' 'p-nu*v2y'} {0 0} ...
                    {'mu-nu*la1x'   '-nu*la1y'} ...
                    {  '-nu*la2x' 'mu-nu*la2y'} {0 0} } };
12 fem.equ.f = { {'-v1d*v1dx-v2d*v1dy' ...
                  '-v1d*v2dx-v2d*v2dy' 'v1dx+v2dy' ...
                  'la1/alpha-v1*v1x-v2*v1y' ...
                  'la2/alpha-v1*v2x-v2*v2y' 'v1x+v2y' ...
                  'J1+v1*la1x+v2*la1y-v1x*la1-v2x*la2' ...
                  'J2+v1*la2x+v2*la2y-v1y*la1-v2y*la2' 'la1x+la2y'}};
15 fem.bnd.r = { {'v1d-g1' 'v2d' 'v1-g1' 'v2' 'la1' 'la2'} ...
                 {'v1d'    'v2d' 'v1'    'v2' 'la1' 'la2'} };
18 fem.sol = femnlin(fem,'solcomp',{'v1d' 'v2d' 'pd'});
19 fem.const = {'nu' '0.001' 'alpha'  '0.01'};
21 fem.xmesh = meshextend(fem);
22 fem.sol = femnlin(fem, ...
           'solcomp',{'v1' 'v2' 'p' 'la1' 'la2' 'mu'},'u',fem.sol);
23 J = postint(fem,'(v1-v1d)*(v1-v1d)+(v2-v2d)*(v2-v2d)')/2 ...
    + postint(fem,'(la1*la1+la2*la2)/alpha')/2;
```

Figure 6: Script for distributed control of a driven cavity flow in 2-D. Lines 1-2, 6-9, 13-14, and 16-17 as in Fig. 1. The desired state is computed in line 18. The optimality system is set up and solved in lines 19-22. Note the repetition of the `meshextend` command. In line 22 the last two additional parameters passed to the solver ensure that the solution component `u` from the `fem.sol` substructure is used in the current computation. The output of a solver is stored in the variable `fem.sol.u`, no matter what names were given to the unknowns in line 3. Here `fem.sol.u` contains the output `v1d`, `v2d`, `pd` from the computation in line 19. Line 23 evaluates the cost functional.

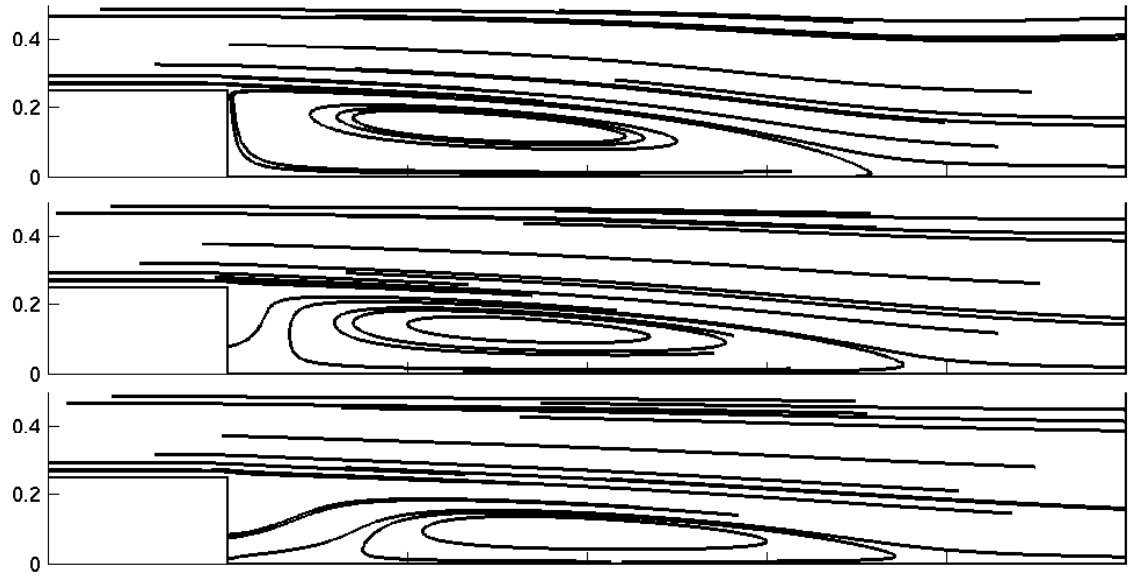

Figure 7: Uncontrolled (top) and controlled (middle: $\alpha = 1$, bottom: $\alpha = 0.1$) backward facing step flow, $\nu = \frac{1}{1000}$. The cost was reduced to 37% and 5.9%, respectively. 3427 pressure nodes, computations with a finer grid (13469 nodes) were also possible on a machine with 1 GByte RAM, and produced similar results.

there is a long separation bubble behind the step. Aim was to reduce this bubble, compare Fig 7. For this purpose we used the cost functional (compare [4])

$$J(v, u_B) \quad = \quad \frac{1}{2} \left\| \begin{bmatrix} \min\{v_1, 0\} \\ \min\{-v_2, 0\} \end{bmatrix} \right\|^2_{L^2(\Omega)^2} + \frac{\alpha}{2} \|u_B\|^2_{L^2(\Sigma_c)^2}. \tag{19}$$

The velocity on the vertical boundary $\Sigma_c := \{0.5\} \times]0, 0.25[$ was used as control parameter. The crucial point is to get access to the normal derivative of the adjoint variable (see (12)) that becomes the boundary condition in the state equations of the optimality system. From (12) and (17) we get

$$u_B \quad = \quad -(\nu \partial_\eta \lambda - \mu \eta)/\alpha \; = \; \frac{1}{\alpha} \begin{bmatrix} -\nu(\lambda_{1x}\eta_1 + \lambda_{1y}\eta_2) + \mu\eta_1 \\ -\nu(\lambda_{2x}\eta_1 + \lambda_{2y}\eta_2) + \mu\eta_2 \end{bmatrix}.$$

Conveniently COMSOL MULTIPHYSICS allows to access the components η_1, η_2 of the outer normal vector via the variables `nx` and `ny` such that u_B can be implemented as boundary condition. The COMSOL MULTIPHYSICS script is shown in Fig. 8. Here no desired state was used, so the corresponding variables and equations are skipped. Fig. 7 shows that the optimization was quite successful for $\alpha = 1$ and 0.1, the computed control is shown in Fig. 9. The convergence was slower than for the driven cavity example, the damped Newton method took 41 iterations from the start (with zero as initial value) for $\alpha = 1$ and additional 17 ones in a restart with $\alpha = 0.1$. It is also possible to increase the stopping criterion for the first iteration (e.g. `'ntol',1e-2` for $\alpha = 1$), and restart from there. Note that $\nu = \frac{1}{1000}$ is quite low for control in this configuration.

6.2 Sequential or iterative approach

As contrast to the one shot approach it is possible to solve the state equation firstly, then the adjoint equation with the just computed state, and perform an update of the control using the Lagrange multiplier and formula (5). This procedure is then iterated until convergence is reached. To avoid the implementation of the optimization algorithm one may use a library, or in the

```
 3 fem.dim = {'v1' 'v2' 'p' 'la1' 'la2' 'mu'};
 4 fem.shape = [2 2 1 2 2 1];
10 fem.equ.expr = {'J1' '-min(v1,0)' 'J2' 'min(-v2,0)'};
11 fem.equ.ga = { { {'p-nu*v1x'  '-nu*v1y'} ...
                    { '-nu*v2x' 'p-nu*v2y'} {0 0} ...
                    {'mu-nu*la1x'   '-nu*la1y'} ...
                    {  '-nu*la2x' 'mu-nu*la2y'} {0 0} } };
12 fem.equ.f = { {'-v1*v1x-v2*v1y' '-v1*v2x-v2*v2y' 'v1x+v2y' ...
                  'J1+v1*la1x+v2*la1y-v1x*la1-v2x*la2' ...
                  'J2+v1*la2x+v2*la2y-v1y*la1-v2y*la2' 'la1x+la2y'}};
15 fem.bnd.r = { {'v1-g1' 'v2' 'la1' 'la2'} {'v1' 'v2' 'la1' 'la2'} {0 0 0 0} ...
                {'v1+(nu*(la1x*nx+la1y*ny)-mu*nx)/alpha' ...
                 'v2+(nu*(la2x*nx+la2y*ny)-mu*ny)/alpha' 'la1' 'la2'} };
17 fem.bnd.g = { {0 0 0 0 0 0} {0 0 0 0 0 0} ...
                {0 0 0 '-v1*la1' '-v1*la2' 0} {0 0 0 0 0 0} };
19 fem.const.nu = 0.001;
20 fem.const.alpha = 1;
21 fem.xmesh = meshextend(fem);
22 fem.sol = femnlin(fem,'maxiter',50,'ntol',1e-4);
23 fem.const.alpha = 0.1;
24 fem.xmesh = meshextend(fem);
25 fem.sol = femnlin(fem,'init',fem.sol,'minstep',1e-8,'ntol',1e-4);
26 J = postint(fem,'min(u,0)*min(u,0)+min(-v,0)*min(-v,0)')/2 ...
     + postint(fem,'((nu*(la1x*nx+la1y*ny)-mu*nx)*(nu*(la1x*nx+la1y*ny)-mu*nx) ...
                    +(nu*(la2x*nx+la2y*ny)-mu*ny)*(nu*(la2x*nx+la2y*ny)-mu*ny))/alpha',...
                   'edim',1,'dl',4)/2;
```

Figure 8: Script for boundary control of a backward facing step. Lines 1, 13-14 are as in Fig. 2, lines 2, 8, 16 as in Fig. 1. Lines 23-25 show a restart with smaller regularization parameter. The option `init` indicates that the previous solution in `fem.sol` is taken as initial guess, `minstep` is the minimal damping factor for Newton's method, `maxiter` the maximum number of steps. Line 26 shows the evaluation of the cost.

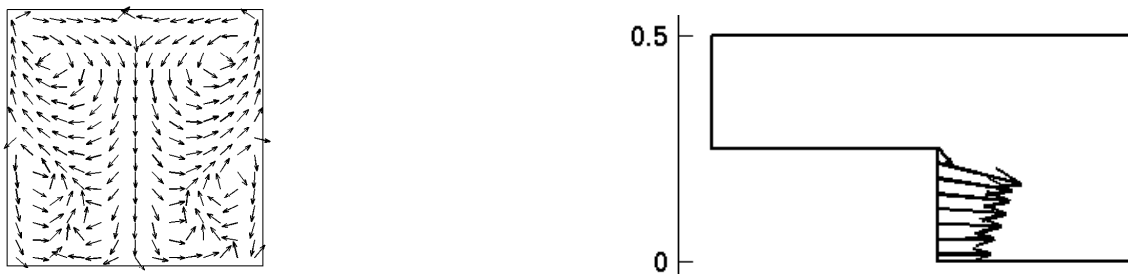

Figure 9: Control for driven cavity (left, $\alpha = 0.01$) and backward facing step ($\alpha = 0.1$), $\nu = \frac{1}{1000}$.

```
 1 global uglob nodes n
 2 fcns{1}.type='inline'; fcns{1}.name='u1(x,y)';
 3 fcns{2}.type='inline'; fcns{2}.name='u2(x,y)';
 4 fem.functions = fcns;
 5 fem.sol = femnlin(fem,'solcomp',{'v1' 'v2' 'p'});
 6 u = posteval(fem,'v1');  nodes = u.p;  n = length(u.d);
 7 options = optimset('GradObj','on');
 8 u = fminunc(@cost,u,options,fem);

 9 function [f,df] = cost(u,fem)
10   global uglob
11   uglob = u;
12   fem.xmesh = meshextend(fem);
13   fem.sol = femnlin(fem,'solcomp',{'v1' 'v2' 'p'});
14   f=postint(fem,'min(v1,0)*min(v1,0)+min(-v2,0)*min(-v2,0)')/2...
      +postint(fem,'(u1(x,y)*u1(x,y)+u2(x,y)*u2(x,y))/alpha')72;
15   fem.sol = femlin(fem,'solcomp',{'la1' 'la2' 'mu'},'u',fem.sol);
16   la1 = posteval(fem,'la1');   la2 = posteval(fem,'la2');
17   df = fem.const.alpha*u-[la1.d,la2.d];

18 function f = u1(x,y)
19   global uglob nodes n
20   f = griddata(nodes(1,:),nodes(2,:),uglob(1:n),x,y);
```

Figure 10: Part of a script for an iterative solution for distributed control. The global copy `uglob` and the storing of the coordinates and number of grid points (line 6) is needed for the functions `u1`, `u2` that realize the control.

MATLAB case, a toolbox routine. Then J and $J'(u)$ have to be implemented, where the former incorporates the solution of the state, the latter in addition the solution of the adjoint equation. The advantage is that the systems to be solved have "only" the size of the state equation, whereas in the one-shot approach the coupled system of double size has to be solved. Moreover different optimization routines may be used, and additional constraints may also be incorporated without changing the underlying COMSOL MULTIPHYSICS setting. We assume that we use an optimization routine that needs the implementation of the cost in a function that takes the current control (say `u`) as parameter, and returns the value of the cost and the gradient (say `J` and `DJ`) as output. This is for example the case for optimization routines in MATLAB's optimization toolbox. An unconstrained minimization using a quasi-Newton method is performed in the script which is partially shown in Fig. 10. Line 7 specifies that the user-defined function `cost` also returns the gradient. The last parameter `fem` for the optimization routine in line 8 is passed to the function `cost` as second parameter. The update formulas (11) or (13) for the control (implemented in line 17 in Fig. 10), it is inevitable to realize the control as a COMSOL MULTIPHYSICS function (see lines 2-4,14,18-20). The efficiency problems that result from this fact have already been discussed in Section 5. For the adjoint equation they could be avoided. This is the main reason that the one shot approach is superior when working with COMSOL MULTIPHYSICS.

6.3 A 3-D example

At the end we briefly show how the 2-D examples can be extended to three space dimensions. The basic proceeding should be clear, and the implementation of the equations for the additional variables `v3`, `la3` is straight forward. An geometry initialization is shown in Fig. 11.

A crucial point in 3-D computations is the necessary storage. We thus just show an example with a rather coarse mesh (864 pressure nodes) and $\nu = 0.01, \alpha = 1$ in Fig. 12. Here boundary control an the vertical wall of the step was used to reduce the cost from (19), with v_2 replaced by v_3.

```
fem.geom = block3(3,1,1,'base','corner','pos',[0 -0.5 -0.5]) ...
         - block3(0.5,1,0.5,'base','corner','pos',[0 -0.5 -0.5]);
fem.bnd.expr = {'g1' '(1-4*y*y)*(1-16*(z-0.25)*(z-0.25))'};
fem.bnd.ind = [1 2 2 2 2 4 3];
```

Figure 11: Geometry initialization for a backward facing step in 3-D.

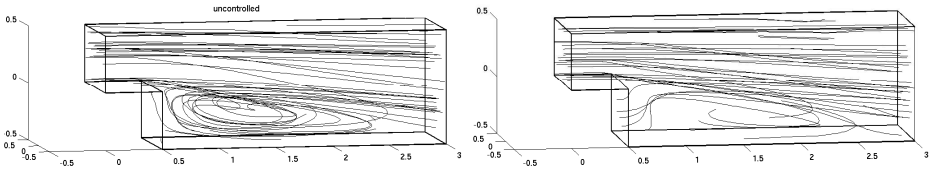

Figure 12: Uncontrolled (left) and controlled backward facing step flow in 3-D for $\nu = \frac{1}{500}$. The results are similar to the 2-D case for high Reynolds number, compare Fig. 7. Only a coarse grid with 470 pressure nodes was used. The cost was reduced to 6.4%.

7 Summary and Outlook

The COMSOL MULTIPHYSICS programming environment is offers the flexibility to solve state and adjoint equations, and thus whole optimality systems for stationary PDE-constrained control problems. The implementation of the adjoint equation can be done with reasonable effort and knowledge of the internal COMSOL MULTIPHYSICS data structure. The effort for this, compared to a complete implementation in another language or environment (as MATLAB), is quite small. The built-in damped Newton solver is rather effective also to solve optimality systems. Due to the internal representation of equations in COMSOL MULTIPHYSICS it turned out to be more appropriate for the control problems than using other optimization routines and coupling them with COMSOL MULTIPHYSICS. The successful application to stationary NSE encourages us to assume that also other nonlinear equations could be considered. Three dimensional problems can be handled, the bottleneck is the needed storage for the direct sparse linear solvers. In 3-D a switch to iterative solvers that are also provided in COMSOL MULTIPHYSICS has to be considered. In principle time-dependent applications are also possible, but they are beyond the scope of this paper.

References

[1] F. Abergel and E. Casas. Some optimal control problems associated to the stationary Navier-Stokes equation. In J.-I. Diaz et al., editor, *Mathematics, climate, and environment, Res. Notes Appl. Math.*, volume 27, pages 213–220. Masson, Paris, 1993.

[2] Comsol Inc. *http://www.femlab.com*.

[3] J.C. de los Reyes and K. Kunisch. A semi-smooth Newton method for control constrained boundary optimal control of the Navier-Stokes equations. *Nonlinear Analysis*, 62:1289–1316, 2005.

[4] M. Desai and K. Ito. Optimal controls of Navier-Stokes equations. *SIAM J. Control Optim.*, 32(5):1428–1446, 1994.

[5] V. Girault and P.-A. Raviart. *Finite Element Methods for Navier-Stokes Equations*. Springer Series in Computational Mathematics 5, New York, 1986.

[6] M.D. Gunzburger, L. Hou, and T.P. Svobodny. Analysis and finite element approximation of optimal control problems for the stationary Navier-Stokes equations with distributed and neumann controls. *Math. Comput.*, 57(195):123–151, 1991.

[7] J.L. Lions. *Optimal Control of Systems Governed by Partial Differential Equations*. Springer, New York, 1971.

[8] The Mathworks, Inc. *http://www.matlab.com*.

[9] J. Nocedal and S.J. Wright. *Numerical Optimization*. Springer, New York, 2000.

[10] T. Roubiček and F. Tröltzsch. Lipschitz stability of optimal controls for the steady state Navier-Stokes equations. *Control and Cybernetics*, 32:683–705, 2003.

[11] T. Slawig. A formula for the derivative with respect to domain variations in Navier-Stokes flow based on an embedding domain method. *SIAM J. Contr. Optim.*, 42(2):495–512, 2003.

[12] R. Temam. *Navier-Stokes Equations*. North-Holland, Amsterdam, 1979.