DFG Research Center MATHEON
Mathematics for key technologies

Strategies for time-dependent PDE control using an integrated modeling and simulation environment. Part one: problems without inequality constraints

Ira Neitzel, Uwe Prüfert, and Thomas Slawig

A MATHEON Preprint

DFG-Research Center MATHEON, Mathematics for key technologies
Technische Universität Berlin, Sekr. MA 3-1
Straße des 17. Juni 136 D-10623 Berlin, Germany

STRATEGIES FOR TIME-DEPENDENT PDE CONTROL USING AN INTEGRATED MODELING AND SIMULATION ENVIRONMENT. PART ONE: PROBLEMS WITHOUT INEQUALITY CONSTRAINTS

IRA NEITZEL *,+, UWE PRÜFERT **,+, AND THOMAS SLAWIG***,++

Abstract. We show how time-dependent optimal control for partial differential equations can be realized in a modern high-level modeling and simulation package. We summarize the general formulation for distributed and boundary control for initial-boundary value problems for parabolic PDEs and derive the optimality system including the adjoint equation. The main difficulty therein is that the latter has to be integrated *backwards* in time. This implies that complicated implementation effort is necessary to couple state and adjoint equations to compute an optimal solution. Furthermore a large amount of computational effort or storage is required to provide the needed information (i.e the trajectories) of the state and adjoint variables. We show how this can be realized in the modeling and simulation package Comsol Multiphysics , taking advantage of built-in discretization, solver and post-processing technologies and thus minimizing the implementation effort. We present two strategies: The treatment of the coupled optimality system in the space-time cylinder, and the iterative approach by sequentially solving state and adjoint system and updating the controls. Numerical examples show the elegance of the implementation and the efficiency of the two strategies.

AMS subject classification: 49K20, 65K10, 65N30

1. Introduction

In this paper we show how time-dependent optimal control problems can be solved using the equation-based modeling and simulation environment Comsol Multiphysics [1]. This software allows a rather easy way to define, discretize and solve stationary and time-dependent PDEs via the finite element method, with a lot of flexibility in mesh generation, choice of ansatz functions, adaptive solver management, and post-processing facilities.

Our main focus is to show how to deal with the reversed integration direction in time for the state and adjoint equations, respectively, that occur in the control of initial value problems for PDEs, see for example [3]. To treat this problem we present two strategies:

- The first one is the classical approach of sequentially solving state and adjoint equation and updating the control in a gradient-based iterative optimization algorithm. Here the main challenge is to realize the reverse time directions, without any low-level data storage or copying, i.e. just using the provided solution routines of the software.
- The second strategy is to interpret the time as an additional "space" dimension, i.e. to solve the whole optimality system in the space-time cylinder by finite elements. Here the main point is the dimension of the discretized system. For the software the PDE is then "stationary", i.e. a boundary value problem. Then the built-in damped Newton solver can be used, and fully coupled space-time adaptivity can be easily applied.

We show how to implement these two approaches in scripts that consist of less than 50 lines. Moreover we emphasize the delicate adjustments and equation settings that have to be done when dealing with the reverse time integration directions mentioned above.

We treat optimal control problems for parabolic PDEs in one and two space dimensions, for distributed and boundary control, and test problems for the heat equation. Here we computed analytical solutions that enable us to show the applicability, flexibility, but also the limits of the two approaches and to characterize and compare them with respect to accuracy and effort. Nevertheless, the scripts

Key words and phrases. Optimal control, finite element method, control using Comsol Multiphysics .
*neitzel@math.tu-berlin.de Supported by the DFG Schwerpunktprogramm SPP 1253
**pruefert@math.tu-berlin.de Supported by the DFG Research Center Matheon.
***ts@numerik.uni-kiel.de
+TU Berlin - Fakultät II Institut für Mathematik, Straße des 17. Juni 136, 10623 Berlin, Germany
++Christian-Albrechts-Universität zu Kiel, Technische Fakultät, Christian-Albrechts-Platz 4, 24118 Kiel, Germany
[1]Comsol Multiphysics is a registered trademark of Comsol Ab

presented in detail below can be taken for problems where analytic solutions are not known, and can be easily modified or extended for other, also nonlinear problems. A similar treatment of the stationary nonlinear Navier-Stokes equations can be found in [4].

The structure of the paper is as follows: In Section 2 we give a general description of PDE constrained control problems with distributed and boundary control, the form of the optimality system, and adjoint equation. This includes some analytical results concerning the existence and uniqueness of optimal controls. In Section 3 we apply these results to our model-problems and obtain the systems of parabolic PDEs belonging to these specific problems.

Section 4 is devoted to COMSOL MULTIPHYSICS 's equation setting and solution procedure for parabolic PDEs. In Section 5 we show how the optimality systems can be realized in the two approaches mentioned above. Here we present the model scripts in detail and show some numerical results. We end the paper by a short summary and outlook.

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 summarize the main results in Section 2.1 and also refer to standard literature, e.g. [1], for further details.

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

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

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, but are considered in a forthcoming paper. The constraint (1) represents a time-dependent PDE given on a time-space domain $Q = (0,T) \times \Omega$ with corresponding boundary conditions on $\Sigma = (0,T) \times \partial\Omega$ and initial data.

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

$$J(y,u) \;=\; \frac{1}{2}\|y - y_d\|_Y^2 + \frac{\kappa}{2}\|u\|_U^2 \tag{2}$$

with regularization parameter $\kappa > 0$. 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 $\frac{\kappa}{2}\|u\|_U^2$. In a similar way one may consider a functional that measures this distance only on the boundary Σ or a part of it. The state equation (1) consists of the PDE, boundary conditions, and initial data. We consider distributed control problems, where the control appears in the PDE, as well as boundary control problems, where the control enters the formulation in the boundary conditions of the following form:

Example for distributed control

$$\begin{aligned} y_t - \Delta y &= u && \text{in } Q \\ \vec{n}\cdot\nabla y + \alpha y &= g && \text{on } \Sigma \\ y(0) &= y_0 && \text{in } \Omega \end{aligned} \tag{3}$$

Example for boundary control

$$\begin{aligned} y_t - \Delta y &= f && \text{in } Q \\ \vec{n}\cdot\nabla y + \alpha y &= \alpha u && \text{on } \Sigma \\ y(0) &= y_0 && \text{in } \Omega \end{aligned} \tag{4}$$

The functions $f \in L^2(Q)$, $g \in L^2(\Sigma)$, and $y_0 \in L^2(\Omega)$ are fixed.

2.1. Analysis of the optimal control problems.

In the following section we derive the mathematical foundation for the optimal control problems (3) and (4). We are concerned with the existence of a unique optimal solution for the model problems as well as the formulation of first order necessary optimality conditions. We point out that due to the convexity of the model problems these conditions are also sufficient for optimality. The following results are well-known in the optimal control community, hence we state them without proofs for the readers' convenience. We rely on the following assumption throughout the remainder of this paper.

Assumption 2.1. *Let $\Omega \subset \mathbb{R}^N$, $N \in \{1,2\}$, be a bounded domain with $C^{0,1}$-boundary Γ if $N = 2$, and a bounded interval in $\mathbb{R}$ if $N = 1$. Moreover, for $Q = \Omega \times (0,T)$, we consider the data $f \in L^2(Q)$, $g \in L^2(\Sigma)$, $\Sigma = \Gamma \times (0,T)$, and $y_0 \in L^2(\Omega)$.*

We further define the solution space

$$W(0,T) = \{y \in L^2(0,T;H^1(\Omega)) \mid y_t \in L^2(0,T,H^1(\Omega)^*)\}.$$

Then a known result by Wloka, [6], or Lions, [1], concerning the solvability of the state equation reads as follows.

Theorem 2.2. *For any triple $(f, g, y_0) \in L^2(Q) \times L^2(\Sigma) \times L^2(\Omega)$ the initial boundary value problem*

$$\begin{aligned} y_t - \Delta y &= f \quad \text{in } Q,\\ \partial_n y + \alpha y &= g \quad \text{on } \Sigma,\\ y(\cdot,0) &= y_0 \quad \text{in } \Omega \end{aligned}$$

admits a unique solution $y \in W(0,T)$.

With the help of Theorem 2.2 we can define the solution operator

$$G = G_Q + G_\Sigma + G_\Omega : L^2(Q) \times L^2(\Sigma) \times L^2(\Omega) \to W(0,T),$$

with

$$G_Q(f) = y(f,0,0), \quad G_\Sigma(g) = y(0,g,0), \quad G_\Omega(y_0) = y(0,0,y_0),$$

where y denotes the unique solution of the parabolic PDE with corresponding data. Hence, we can express the dependence $y = y(u)$ with the help of the solution operator G. We point out that the given fixed data f, g, and y_0 do not influence the optimization process. For simplicity, we assume them to be zero and use the abbreviated notation $y = Gu$, where $G := G_Q$ or $G = G_\Sigma$ for distributed and boundary control, respectively. Consequently, the reduced objective function $\hat{J}$ reads $\hat{J}(u) = J(Gu,u)$. Due to the fact that $U = L^2(Q)$ or $U = L^2(\Sigma)$, respectively, are not empty, the following theorem holds by standard arguments.

Theorem 2.3. *Under Assumption 2.1, the optimal control problems* (3) *and* (4) *admit for each $\kappa > 0$ unique optimal controls $u^* \in L^2(Q)$ or $u^* \in L^2(\Sigma)$, respectively.*

In the following theorems we formulate the first order necessary optimality conditions for (3) and (4). A more detailed explanation how these conditions can formally be derived will be given in the next section.

Theorem 2.4. *Let u^* be the optimal control of the distributed control problem* (3) *with associated state y^*. Then there exists an adjoint state $p \in W(0,T)$ such that p is the weak solution of*

$$\begin{aligned} -p_t - \Delta p &= y^* - y_d \quad \text{in } Q,\\ \partial_n p + \alpha p &= 0 \qquad\quad\ \text{on } \Sigma,\\ p(\cdot,T) &= 0 \qquad\quad\ \text{in } \Omega, \end{aligned} \tag{5}$$

and the gradient equation

$$\kappa(u^* - u_d) + p = 0$$

is fulfilled for almost all $(x,t) \in Q$. Analogously, let u^* be the optimal solution of the boundary control problem (4)*. Then there exists an adjoint state $p \in W(0,T)$ satisfying the same adjoint equation, as well as the gradient equation*

$$\kappa(u^* - u_d) + p = 0$$

for almost all $(x,t) \in \Sigma$.

We refer to [5] or [1] for further details. We point out that the adjoint equation can be transformed into an initial-boundary value problem by the time transformation $\tau = T - t$, which allows to employ Theorem 2.2 for the existence of a unique solution of (5) for given right-hand-sides.

2.2. **Adjoint equations and optimality systems.** In this subsection we show the general process of obtaining the optimality system or a representation of the gradient of $\hat{J}$. We introduce the Lagrange functional associated with the constrained problem

$$\min_{u\in U} J(y,u) \quad \text{subject to} \quad (1). \tag{6}$$

It is defined as

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

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 and initial values) we have $Z = Z_Q \times Z_\Sigma \times Z_\Omega$. Thus Z^* is isometrically isomorphic to the Cartesian product of the duals i.e. $Z^* \cong Z_Q^* \times Z_\Sigma^* \times Z_\Omega^*$, and the duality pairing is given as

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

where i is the i-th element of $\{Q,\Sigma,\Omega\}$. The necessary optimality condition for a saddle point of L and a minimum of (6) are computed by setting the directional derivatives of L with respect to (y,u,z) equal to zero in all admissible directions $(\bar{y},\bar{u},\bar{\lambda})$. We arrive at

$$\begin{array}{rclcl} L_y(y,u,\lambda)\bar{y} &=& J_y(y,u)\bar{y} + \langle \lambda, e_y(y,u)\bar{y}\rangle_{Z^*,Z} &=& 0 \quad \forall \bar{y}\in Y \\ L_u(y,u,\lambda)\bar{u} &=& J_u(y,u)\bar{u} + \langle \lambda, e_u(y,u)\bar{u}\rangle_{Z^*,Z} &=& 0 \quad \forall \bar{u}\in U \\ L_\lambda(y,u,\lambda)\bar{\lambda} &=& \langle \bar{\lambda}, e(y,u)\rangle_{Z^*,Z} &=& 0 \quad \forall \bar{\lambda}\in Z^*, \end{array} \tag{7}$$

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

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

where A^* denotes the adjoint operator of A. The optimality system (7) may be solved directly in the so-called *one-shot-approach*. An alternative approach is to use a gradient-based iterative algorithm to solve (6). Then the directional derivative of $\hat{J}$ is needed. By the chain rule it is given as

$$\hat{J}'(u)\bar{u} \;=\; J_y'(y,u)y'(u)\bar{u} + J_u'(y,u)\bar{u}. \tag{8}$$

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

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

Adding (8) and the duality pairing between this expression and the Lagrange multiplier λ gives

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

The first two terms result in 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 solving both the state and the adjoint equation

$$\hat{J}'(u)\bar{u} \;=\; J_u'(y,u)\bar{u} + \langle \lambda, e_u'(y,u)\bar{u}\rangle_{Z^*,Z} \tag{9}$$

can be evaluated and used in an iterative optimization algorithm.

3. Application to the examples

Now we specialize the results of the last section for distributed and boundary control for our model-problems described in Section 2.

3.1. **Lagrange functional, adjoint equation, and optimality systems.** In this paper we consider only problems with distributed observation, i.e. the distance between the state y and the desired state y_d is measured over the whole space-time domain. A possible extension to other types of objective functions depends strongly on their properties, e.g. differentiability, etc. Generally, we will denote the optimal control by u^* with assocoated optimal state y^*.

3.1.1. *Distributed control.* The objective functional has the form

$$J(y,u) = \frac{1}{2}\|y-y_d\|_Q^2 + \frac{\kappa}{2}\|u-u_d\|_Q^2. \tag{10}$$

The function u_d can be interpreted as a desired control, in many applications it will be zero. However, it is helpful to construct problems with known exact solutions.

The constraint e is given by $e(y,u) = \sum_{i=1}^{n} e_i(y,u)$ with

$$e_i(y,u) \quad = \quad \begin{cases} y_t - \Delta y - u & \text{in } Q, \quad i=1 \\ \vec{n}\cdot\nabla y + \alpha y - g & \text{on } \Sigma, \quad i=2 \\ y(0) - y_0 & \text{in } \Omega, \quad i=3 \end{cases}$$

The Lagrangian has the form

$$\begin{aligned} L(y,u,p) = J(y,u) &- (y_t - \Delta y - u, p)_Q \\ &- (\vec{n}\cdot\nabla y + \alpha y - g, p)_\Sigma - (y(0)-y_0, p)_\Omega. \end{aligned}$$

Here, J is the cost-functional (10), $Y = L^2(Q)$ and $U = L^2(Q)$. In a standard way, we obtain the adjoint equation by computing the derivative in direction $\bar{y}$, cf. the first equation in (7),

$$\begin{aligned} -p_t - \Delta p &= y^* - y_d \ \text{ in } Q \\ \vec{n}\cdot\nabla p + \alpha p &= 0 \qquad\quad \text{on } \Sigma \\ p(T) &= 0. \qquad\quad \text{in } \Omega \end{aligned}$$

The gradient equation is given by the derivative of L with respect to u. We get

$$\kappa\,(u^* - u_d) + p = 0 \quad \text{in } Q. \tag{11}$$

At last, the state equation

$$\begin{aligned} y_t^* - \Delta y^* &= u^* \ \text{ in } Q \\ \vec{n}\cdot\nabla y^* + \alpha y^* &= g \ \text{ on } \Sigma \\ y^*(0) &= y_0 \ \text{ in } \Omega \end{aligned} \tag{12}$$

has to be fulfilled.

3.1.2. *Boundary control.* The objective functional reads

$$J(y,u) = \frac{1}{2}\|y-y_d\|_Q^2 + \frac{\kappa}{2}\|u-u_d\|_\Sigma^2. \tag{13}$$

In this case, the function e is given by

$$e_i(y,u) \quad = \quad \begin{cases} y_t - \Delta y - f & \text{in } Q, \quad i=1 \\ \vec{n}\nabla y + \alpha y - u & \text{on } \Sigma, \quad i=2 \\ y(0) - y_0 & \text{in } \Omega, \quad i=3. \end{cases}$$

and the Lagrangian function reads here as

$$\begin{aligned} L(y,u,p) = J(y,u) &- (y_t - \Delta y - f, p)_Q \\ &- (\vec{n}\cdot\nabla y + \alpha(y-u), p)_\Sigma - (y(0)-y_0, p)_\Omega, \end{aligned}$$

which gives us the same adjoint equation as above. However, the gradient equation holds on the boundary Σ:

$$\kappa\,(u^* - u_d) + p = 0 \quad \text{on } \Sigma.$$

The state equation is here given by

$$\begin{aligned} y_t^* - \Delta y^* &= f \qquad \text{in } Q \\ \vec{n}\cdot\nabla y^* + \alpha y^* &= u^* \quad \text{on } \Sigma \\ y^*(0) &= y_0 \qquad \text{in } \Omega. \end{aligned}$$

In all cases, we have to solve a coupled system of two time-dependent PDEs for the state y and the adjoint state p, and the algebraic gradient equation, which implements the coupling between the adjoint and the control. By uniqueness of y, p, and u, we can write for all $\kappa > 0$ the control u as $u = -\frac{1}{\kappa}p + u_d$ and replace u by this term in the state equation. Now, we only have to solve the adjoint equation and the modified state equation, a coupled system of two time-dependent PDEs.

Unfortunately, the adjoint equation is a so called backward-in-time equation (with initial value given at end-time T) where the state equation is a common forward-in-time equation. Both equations are well posed, but the challenge lies in solving this system numerically.

4. Solving time dependent PDEs with 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 order from one to four. It provides a graphical user interface as well as a scripting mode. Doing PDE-constrained control, we have to define the adjoint equation and implement the coupling between state, adjoint and control. But since Comsol Multiphysics allows the user to define an almost arbitrary PDE, all its solution and post-processing features can be used also to solve optimality systems.

In this section we describe at first two possible approaches to solve coupled time-dependent problems: a one-shot-approach, which uses space-time discretization and an iterative approach, which uses the capability of Comsol Multiphysics to solve backward-in-time problems "out-of-the-box". To become familiar with the syntax of Comsol Multiphysics , we first demonstrate how to solve a single PDE. Throughout, we will use the so called scripting mode of Comsol Multiphysics .

4.1. **Time-space-elements.** Comsol Multiphysics provides no space-time elements in the sense of Vexler et. al., see for details [2]. Our idea is to define the time as an additional space dimension. In this sense, we consider the space-time domain Q where space and time are treated the same.
One example: Solving Burgers equation. Often, Burgers equation is used as a benchmark-problem for numerical algorithms. Let $x \in \Omega = (0, \pi)$ a space and $t \in (0, T)$ a time interval. We consider the problem of finding $y \in Y$ such that y fulfills the time dependent, nonlinear PDE

$$\begin{aligned} y_t - y_{xx} + y_x y &= 0 && \text{in } \Omega \times (0,T) \\ y &= 0 && \text{on } \Sigma \\ y(0) &= y_0 && \text{in } \Omega. \end{aligned}$$

The first row is equivalent to $y_t - \frac{d}{dx}\left(\frac{dy}{dx} + \frac{1}{2}y^2\right)$. We choose $y_0 = \sin(2x)$ and solve the problem on the time-interval $(0, 5)$. The following code sequence solves the problem by using linear space-time elements on an adaptively refined triangle-mesh.

```
% geometry and mesh:
(1)  fem.geom = rect2(0,pi,0,5);
(2)  fem.mesh = meshinit(fem,'hauto',3);
(3)  fem.sdim = {'x' 'time'};
% unknowns and element types:
(4)  fem.dim = {'y'};
(5)  fem.shape = [1];
% parameters:
(6)  fem.const = {'nu' '1e-2'};
(7)  fcns{1}.type='inline';
(8)  fcns{1}.name='y_0(x)';
(9)  fcns{1}.expr='sin(2*x)';
(10) fem.functions = fcns;
% pde-form
(11) fem.form = 'general';
% coefficients + rhd side:
(12) fem.equ.ga = { { {'-nu*yx+0.5*y^2' '0' } }  };
(13) fem.equ.f = { {'-ytime'} };
% boundaries: 1:t=0,2:x=pi,3:t=5,4:x=0
(14) fem.bnd.ind = [1 2 3 2];
% boundary conditions:
(15) fem.bnd.r = {{'y-y_0(x)'};
                 {'y' };
                 { 0 }};
```

```
(16) fem.bnd.g = {{0};
                  {0};
                  {0} };

(17) fem.xmesh = meshextend(fem);
% solve:
(18) fem = adaption(fem,'ngen',2,'Maxiter',50,'Hnlin','off');
% plotting the adaptive mesh and the solution
(19) subplot(121),
(20)  meshplot(fem);
(21)  subplot(122),
(22)  postplot(fem,'tridata','y','triz','y','title','y')
```

LISTING 1.

4.1.1. *Geometry and mesh (line 1–3).* Defining the geometry is the first step, here we define a (2D) rectangle $[0,\pi] \times [0,5]$. The mesh initialization follows. The optional parameter `hauto` controls the mesh size (from `1`: fine to `9`: coarse). The first dimension is defined as the space x, the second as *time*.

4.1.2. *Definition of the unknown and the associated finite element type (lines 4–5):* The ansatz function type and order have to be specified:

- We want the unknown to be named `y` instead of the default name `u1`.
- We choose linear finite elements.

4.1.3. *Definition of the constants (lines 6–10).* We have to define the initial value $y(0)$ and the viscosity coefficient ν:

- Setting ν as constant with value 10^{-2}.
- Defining $y(0)$ as an in-line-function. $y(0) = \sin(\pi x)$.

4.1.4. *Definition of the PDE (lines 11–17).* First of all, the form of the equation has to be set (line 11). COMSOL MULTIPHYSICS offers 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 requires the user to write the equation in the weak form.

We use the `general` form. The PDE coefficients have to be defined in a substructure named `fem.equ` of the main structure `fem`. The general equation for time-independent problems reads

$$\nabla \cdot \Gamma = F \quad \text{in } \Omega.$$

In our case, treating time as an additional space dimension, ∇ is the formal operator $\left(\frac{d}{dx}\ \frac{d}{dtime}\right)$. Thus, we have to define $\Gamma = \left(-\frac{dy}{dx} + \frac{1}{2}y^2 \quad 0\right)$ and $F = -\frac{dy}{dtime}$. The PDE-coefficients for Γ are stored in a structure `fem.eq.ga` (line 12), and F is stored in `fem.equ.f` (line 13).

4.1.5. *Boundary conditions.* The substructure `fem.bnd` defines the boundary conditions. Based on the initial geometry, different boundary sections can be defined (line 14).

Our problem has three different types of boundary conditions

- homogeneous Dirichlet boundary conditions on Σ (boundaries nos. two and four).
- inhomogeneous Dirichlet boundary conditions on $\Omega \times \{0\}$: $y(0) = \sin(\pi x)$ (boundary no. one)
- no boundary condition on $\Omega \times \{5\}$, since this corresponds to the end of the time interval (boundary no. three).

The numbering of the edges has to be found by trial and error. The function `edgelabel` is helpful to find the correct ones. Two arrays with the edge indices of the two boundary sections have to be put in the cell array `fem.bnd.ind`.

The boundary conditions on each boundary section are written as

$$\left.\begin{aligned} -\vec{n}\cdot\Gamma_l &= G_l + \sum_{m=1}^{M} \frac{\partial R_m}{\partial u_l}\mu_m, & l &= 1,\dots,N \\ R_m &= 0, & m &= 1,\dots,M \end{aligned}\right\} \quad \text{on } \partial\Omega. \tag{14}$$

The variables 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 is set to zero. Then the first equation in (14) imposes no condition on the unknowns because of the free Lagrange multipliers. In line 15, we have defined the Dirichlet boundary conditions in the first two entries of the structure `fem.bnd.r`. The corresponding entries in `fem.bnd.g` are set to zero. For the boundary part $\Omega \times \{5\}$ we set `fem.bnd.r` **and** `fem.bnd.g` to zero.

4.1.6. *Assembly and solution of the discrete system (lines 17 and 18).* 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 adaptive solver (line 18) is a damped Newton method and has a wide range of parameters, here we set the number of new (grid) generations `ngen` to two, the maximum number of Newton steps to 50 and the range of the damping parameter near one (`Hnlin`).

4.1.7. *Postprocessing (lines 19–22).* To visualize the results we use the `meshplot` and `postplot` routines of Comsol Multiphysics.

For more details on the used commands and further options we refer to the Comsol Multiphysics documentation.

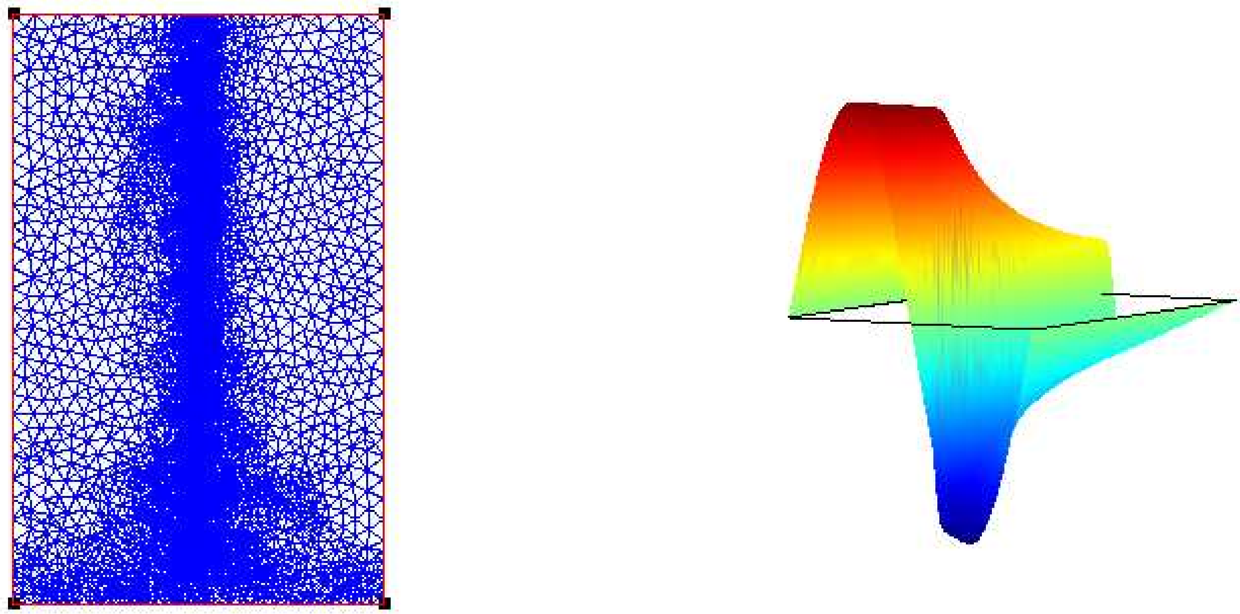


Figure 1. Adaptively refined mesh and solution of Burger equation.

5. Control using Comsol Multiphysics

5.1. **One-shot-approach.** By the term "One-shot-approach" we refer the solution of an optimal control problem by solving the optimality conditions, in most cases a system of possibly time dependent and/or non-linear PDEs, at once.

5.1.1. *An example in 1D.* Our first example is a problem with distributed control.

$$\min J(y,u) = \|y - y_d\|_Q^2 + \frac{\kappa}{2}\|u - u_d\|_Q^2$$

where the pair (y,u) solves the parabolic PDE

$$\begin{aligned} y_t(x,t) - \Delta y(x,t) &= u(x,t) && \text{in } Q \\ \vec{n}\cdot\nabla y(x,t) &= -\sin(t)\vec{n} && \text{on } \Sigma \\ y(x,0) &= 0 && \text{on } \Omega. \end{aligned}$$

The space-time domain Q is $Q = (0,\pi)\times(0,\pi)$. The desired state is given by $y_d(x,t) = \sin(x)\sin(t) - \cos(x) - \cos(x)(\pi - t)$, and the desired control is given by $u_d(t) = \sin(x)(\sin(t)+\cos(t)) + \frac{1}{\kappa}\cos(x)(\pi - t)$.

One can easily check that $y^* = \sin(t)\sin(x)$, $u^* = \sin(t)\sin(x) + \cos(t)\sin(x)$, and $p = -\kappa(u^* - u_d)$ solves the optimality system given in Section 3.1.1.

Implementation of the optimality system. To solve the optimality system, we have to implement a system of PDEs. The adjoint equation (5) can be implemented in a similar way as the state equation. For the implementation of a single PDE in COMSOL MULTIPHYSICS we refer to Section 4.1. The only difference is that in the adjoint equation the time runs backwards and the initial data is given on $t = T$. Listing 2 provides the COMSOL MULTIPHYSICS code solving problem 1. We choose the regularization parameter $\kappa = 1$.

We only explain here the differences to the solution of a single PDE in Listing 1. In line 2, we discretize space and time by a rectangular grid. We use the control-reduced form by setting $u = u_d - \frac{1}{\kappa}p$, which results in a system only in y and p, lines 5 and 18. In line 19 and 20, we define the optimality system

$$\left(\frac{d}{dx}\ \frac{d}{dt}\right)\begin{pmatrix} -\frac{dy}{dx} & -\frac{dp}{dx} \\ 0 & 0 \end{pmatrix} = \begin{pmatrix} -\frac{dy}{dt} - \frac{1}{\kappa}p \\ \frac{dp}{dt} + y - y_d \end{pmatrix}. \tag{15}$$

The syntax of COMSOL MULTIPHYSICS allows us to write the columns of the lhs-matrix in (15) as rows in the cell-array `fem.equ.ga`. In line 20, COMSOL MULTIPHYSICS substitutes $u_d - \frac{1}{\kappa}p$ by u. Again, the column of the rhs in (15) will be written as a row in the cell-array `fem.equ.f`. The Dirichlet boundary conditions on boundaries nos. one and three implements the initial values $y - y_0 = 0$ and $p(T) = 0$ (line 22). The Neumann part of the boundary-value definition in `fem.bnd.g` is set to $\vec{n}\cdot\nabla y = -\sin(t)$ and $\vec{n}\cdot\nabla p = 0$ (line 23). In line 25 we call the linear solver `femlin`.

```
(1)   fem.geom = rect2(0,pi,0,pi);
(2)   fem.mesh = meshinit(fem,'hmax',2^(-3));
(3)   fem.sdim = {'x' 'time'};
% unknowns and element types:
(4)   fem.form = 'general';
(5)   fem.dim = {'y' 'p'};
(6)   fem.shape = [2 2];
% parameters:
(7)   fem.const = {'kappa' '0.1'};
(8)   fcns{1}.type='inline';
(9)   fcns{1}.name='ud(x,time)';
(10)  fcns{1}.expr='sin(x)*(sin(time)+cos(time))+cos(x)*(pi-time)';
(11)  fcns{2}.type='inline';
(12)  fcns{2}.name='yd(x,time)';
(13)  fcns{2}.expr='sin(x)*sin(time)-cos(x)-cos(x)*(pi-time)';
(14)  fcns{3}.type='inline';
(15)  fcns{3}.name='g(time)';
(16)  fcns{3}.expr='-sin(time)';
(17)  fem.functions = fcns;
% coefficients + rhd side:
(18)  fem.globalexpr = {'y0' '0' 'u' 'ud(x,time)-p/kappa'};
(19)  fem.equ.ga = { { {'-yx' '0'} {'-px' '0'} } };
(20)  fem.equ.f = { {'-ytime+u' 'ptime+y-yd(x,time)'} };
```

```
% boundaries: 1:t=0,2:x=pi,3:t=pi,4:x=0
(21)  fem.bnd.ind = [1 2 3 2];
% boundary conditions:
(22)  fem.bnd.r = { {'y-y0' 0};
                    {0 0};
                    {0 'p'} };
(23)  fem.bnd.g = { {0 0};
                    {'g(time)' '0'};
                    {0 0} };
(24)  fem.xmesh = meshextend(fem);
% solve:
(25)  fem.sol = femlin(fem);
% postprocessing
(26)  subplot(221),meshplot(fem)
(27)  subplot(222),postplot(fem,'tridata','y','triz','y','title','y')
(28)  subplot(223),postplot(fem,'tridata','u','triz','u','title','u')
(29)  subplot(224),postplot(fem,'tridata','p','triz','p','title','p')
```

LISTING 2.

Results. Table 1 shows the L^2-errors $\|\bar{u}-u_h\|_Q$ and $\|\bar{y}-y_h\|_Q$ depending on the mesh size parameter `hmax` .

h_{max}	$\|u^* - u_d\|_Q$	$\|y^* - y_h\|_Q$	$\|J(y^*,u^*) - J(y_h,u_h)\|$
2^{-2}	0.016417	0.00032837	0.0011716
2^{-3}	0.0022293	$3.079 \cdot 10^{-5}$	$8.3556 \cdot 10^{-5}$
2^{-4}	0.00030615	$5.0814 \cdot 10^{-6}$	$1.025 \cdot 10^{-5}$
2^{-5}	$4.0305 \cdot 10^{-5}$	$4.9791 \cdot 10^{-7}$	$4.7661 \cdot 10^{-7}$
2^{-6}	$5.373 \cdot 10^{-6}$	$5.9155 \cdot 10^{-8}$	$6.6537 \cdot 10^{-8}$

TABLE 1. Errors $\|\bar{u}-u_h\|_Q$ and $\|\bar{y}-y_h\|_Q$.

h_{max}	$\|u^* - u_d\|_Q$	$\|y^* - y_h\|_Q$	$\|J(y^*,u^*) - J(y_h,u_h)\|$
2^{-3}	0.00036247	$5.2636 \cdot 10^{-6}$	$4.9854 \cdot 10^{-6}$
2^{-4}	$4.9127 \cdot 10^{-5}$	$8.0412 \cdot 10^{-7}$	$4.7442 \cdot 10^{-7}$
2^{-5}	$6.6022 \cdot 10^{-6}$	$9.3921 \cdot 10^{-8}$	$5.787 \cdot 10^{-8}$

TABLE 2. Errors $\|u^*-u_h\|_Q$ and $\|y^*-y_h\|_Q$ adaptive solver, two new grid-generations, started at h_{max}.

5.1.2. *Boundary control.* We consider the following problem:

$$\min J(y,u) = \|y-y_d\|_Q^2 + \frac{\kappa}{2}\|u-u_d\|_\Sigma^2$$

where the pair (y,u) solves the parabolic PDE

$$\begin{aligned} y_t(x,t) - \Delta y(x,t) &= f(x,t) && \text{in } Q \\ \vec{n}\cdot\nabla y(x,t) &= u(x,t) && \text{on } \Sigma \\ y(x,0) &= 0 && \text{on } \Omega. \end{aligned}$$

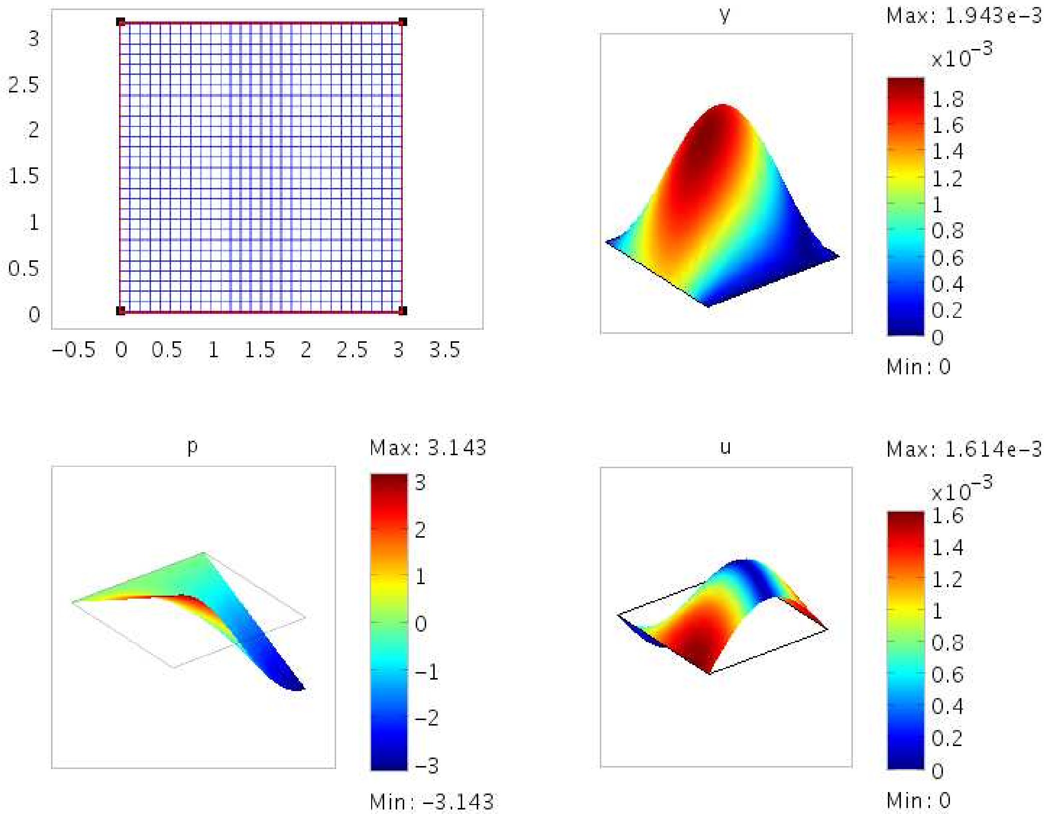


FIGURE 2. Mesh and solutions to example 1. `hauto` set to 3.
The colors in the plot of the state and control indicate the error.

The domain Q is given as in Example 2.

The functions $u_d(x,t) = -\sin(t) + \frac{1}{\kappa}(\pi - t)\vec{n}$, $y_d(x,t) = \sin(x)\sin(t) - \cos(x) - cos(x)(\pi - t)$ and $f(x,t) = \sin(x)\cos(t) + \sin(x)\sin(t)$ are given, and κ is set to 10^{-2}. For the details of the implementation of these functions, see Listing 3.

```
fem.const = {'alpha' '0' 'kappa' '0.01'};
fcns{1}.type='inline'; fcns{1}.name='ud1(x,time)';
fcns{1}.expr='-sin(time)+100*(pi-time)';
fcns{2}.type='inline'; fcns{2}.name='ud2(x,time)';
fcns{2}.expr='-sin(time)+100*(time-pi)';
fcns{3}.type='inline'; fcns{3}.name='yd(x,time)';
fcns{3}.expr='sin(x)*sin(time)-cos(x)-cos(x)*(pi-time)';
fcns{4}.type='inline'; fcns{4}.name='f(x,time)';
fcns{4}.expr='sin(x)*cos(time)+sin(x)*sin(time)';
```

LISTING 3.

The optimal control is given by $u^* = -\sin(t)$. The control is implemented by defining the boundary conditions in `fem.bnd.r` and `fem.bnd.r`, see Listing 4.

```
fem.bnd.r = { {'y-y0' 0} {0 0} {0 'p'} {0 0} };
fem.bnd.g = { {0 0}
              {'ud2(x,time)-p' '0'}
              {0 0}
              {'ud1(x,time)-p' '0'}};
```

LISTING 4.

Again, we use $u = \frac{1}{\kappa}p - u_d$ to substitute u by p. The rest of the COMSOL MULTIPHYSICS-code is the same as for Example 5.1.1.

Results. In the following tables we present the errors for the control, the state, and the values of the objective function depending on the `meshsize` h_{max} of the space-time mesh. Table 3 presents the results computed by the linear solver, while Table 4 shows the errors resulting by solving the problem with the adaptive solver, starting at a mesh generated with h_{max} as shown in the first column.

h_{max}	$\|u^* - u_d\|_Q$	$\|y^* - y_h\|_Q$	$\|J(y^*, u^*) - J(y_h, u_h)\|$
2^{-2}	0.21379	0.025347	0.17679
2^{-3}	0.10525	0.009645	0.025116
2^{-4}	0.043199	0.0029205	0.0018136
2^{-5}	0.017076	$7.7394 \cdot 10^{-4}$	$2.3996 \cdot 10^{-4}$
2^{-6}	0.0057762	$1.9415 \cdot 10^{-4}$	$2.1888 \cdot 10^{-4}$

TABLE 3. Errors for the boundary control problem.

h_{max}	$\|u^* - u_d\|_Q$	$\|y^* - y_h\|_Q$	$\|J(y^*, u^*) - J(y_h, u_h)\|$
2^{-3}	0.04445	0.002287	0.001777
2^{-4}	0.015699	$6.8792 \cdot 10^{-4}$	$1.4786 \cdot 10^{-4}$
2^{-5}	0.0053319	$1.4277 \cdot 10^{-4}$	$6.6657 \cdot 10^{-5}$

TABLE 4. Errors $\|u^* - u_h\|_Q$ and $\|y^* - y_h\|_Q$ adaptive solver, two new grid-generations, started at h_{max}.

5.1.3. *An example in 2D.* In this example we use the 3D capability of COMSOL MULTIPHYSICS to solve a problem in two space dimensions. Consider the optimal control problem

$$\min J(y,u) = \frac{1}{2}\|y - y_d\|_Q^2 + \frac{1}{2}\|u - u_d\|_Q^2$$

where $(y,u) \in W(0,T)$ are solutions of the parabolic PDE

$$\begin{aligned} y_t - \Delta y &= u \\ \vec{n} \cdot \nabla y &= -\vec{n} \cdot \sin(t)(\sin(x_1), \sin(x_2))^\top \\ y(0) &= 0. \end{aligned}$$

The space-time domain is defined by $Q = (0,\pi)^2 \times (0,\pi) \subset \mathbb{R}^3$. The functions y_d and u_d are given by

$$y_d = \sin(x_1)\sin(x_2)\sin(t) - \cos(x_1)\cos(x_2) - 2\cos(x_1)\cos(x_2)(\pi - t)$$

and

$$u_d = \sin(x_1)\sin(x_2)\cos(t) + 2\sin(x_1)\sin(x_2)\sin(t) + \frac{1}{\kappa}\cos(x_1)\cos(x_2)(\pi - t)$$

resp. The optimal solutions are

$$\begin{aligned} y^*(x_1,x_2,t) &= \sin(x_1)\sin(x_2)\sin(t) \\ u^*(x_1,x_2,t) &= \sin(x_1)\sin(x_2)(\cos(t) + 2\sin(t)) \\ p^*(x_1,x_2,t) &= \cos(x_1)\cos(x_2)(\pi - t), \end{aligned}$$

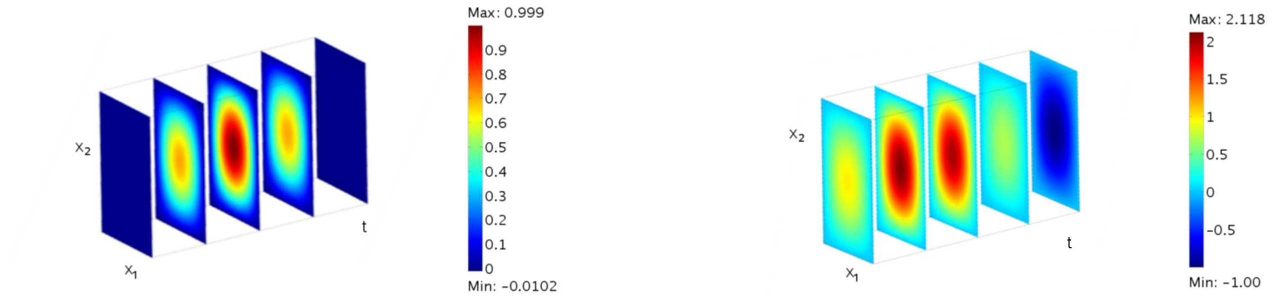
which can easily be checked by inserting them into the optimality conditions (5)–(12). In the following listing, we present the COMSOL MULTIPHYSICS code that solves this problem by the direct method using a discretization of the space-time domain by tetraeders.

```
(1)  clear all
% geometry and mesh:
(2)  fem.geom = block3(pi,pi,pi,'base','corner','pos',[0 0 0]);
(3)  fem.mesh = meshinit(fem,'hauto',5);
(4)  subplot(221),meshplot(fem)
(5)  fem.sdim = {'x1' 'x2' 'time'};
% unknowns and element types:
(6)  fem.form = 'general';
(7)  fem.dim = {'y' 'p'};
(8)  fem.shape = [2 2];
% parameters:
(9)  fem.const = {'alpha' '0' 'nu' '1.e-0'};
(10) fcns{1}.type='inline'; fcns{1}.name='ud(x1,x2,time)';
(11) fcns{1}.expr='sin(x1)*sin(x2)*cos(time)+2*sin(x1)*sin(x2)...
                  *sin(time)-cos(x1)*cos(x2)*(pi-time)';
(12) fcns{2}.type='inline'; fcns{2}.name='yd(x1,x2,time)';
(13) fcns{2}.expr='sin(x1)*sin(x2)*sin(time)-cos(x1)*cos(x2)...
                  -2*cos(x1)*cos(x2)*(pi-time)';
(14) fcns{3}.type='inline'; fcns{3}.name='g1(x1,time)';
(15) fcns{3}.expr='-sin(x1)*sin(time)';
(16) fcns{4}.type='inline'; fcns{4}.name='g2(x2,time)';
(17) fcns{4}.expr='-sin(x2)*sin(time)';
(18) fem.functions = fcns;
% coefficients + rhd side:
(19) fem.globalexpr = {'y0' '0' 'u' 'ud(x1,x2,time)-p/nu'};
(20) fem.equ.ga = { { {'-yx1' '-yx2' '0'} {'-px1' '-px2' '0'} } };
(21) fem.equ.f = { {'-ytime+u' 'ptime+y-yd(x1,x2,time)'} };
% boundaries: 1:t=0,2:t=pi,3:x2=0,4:x2=pi
(22) fem.bnd.ind = [1 4 3 2 4 3];
% boundary conditions:
(23) fem.bnd.r = {  {'y-y0' 0} {0 'p'} {0 0} {0 0} };
(24) fem.bnd.g = {  {0 0} {0 0} {'g1(x1,time)-alpha*y' '-alpha*p'}...
                   {'g2(x2,time)-alpha*y' '-alpha*p'} };
(25) fem.xmesh = meshextend(fem);
% solve:
(26) fem = adaption(fem,'ngen',3);
(27) subplot(221);meshplot(fem);
(28) subplot(222); postplot(fem,'slicedata','y','slicezspacing',...
     [0 pi/4 pi/2 3*pi/4 pi],'slicexspacing',0,'sliceyspacing',0);
(29) subplot(223),postplot(fem,'slicedata','u','slicezspacing',...
     [0 pi/4 pi/2 3*pi/4 pi],'slicexspacing',0,'sliceyspacing',0);
(30) subplot(224),postplot(fem,'slicedata','p','slicezspacing',...
     [0 pi/4 pi/2 3*pi/4 pi] ,'slicexspacing',0,'sliceyspacing',0);
```

Listing 5.

In the following, we point out some differences to the code of the 1D example. In line 2 we define the space-time domain as a (3D) block object. In lines 14–17 we define the outward normal derivative for the boundaries nos. 3 and 4. In line 22 we group the boundaries into four boundary-groups. The first group contains only boundary number one, the time-slice at time $t = 0$. The second group contains only boundary number four, the time-slice at time $t = \pi$. The other boundaries are boundaries in space. The bounds number two and five belong to group number four, while boundaries number three and six belong to group number three. Lines 24 and 25 define the boundary conditions on these boundary groups. In line 27 we use the adaptive solver, and finally, in lines 29–30 we use the `postplot` routine to produce the sliceplots shown in Figure 3.

Results. Again, we tested both solvers, the linear solver `femlin` and the adaptive solver `adaption`. In 3D, the space-time grid consists of tetraeders and the number of unknowns grows cubically when refining the grid. For that reason, we restrict our survey to three grids generated using `meshinit` where refinement is controlled by using the parameter **hauto** ranging from 7 to 4. In Figure 3 we present time-slice plots of u_h, y_h and p_h.



(a) (b)

FIGURE 3. Time-slices of the solution of the 2D example: state y (a) and control u (b). The colors indicate the values of y_h and u_h at the times $t_k = k\pi/4$ with $k = 0, ..., 4$.

hauto	$\|u^* - u_d\|_Q$	$\|y^* - y_h\|_Q$	$\lvert J(y^*, u^*) - J(y_h, u_h)\rvert$
6	0.27129	0.0035163	0.013943
5	0.079391	0.00087821	0.0032201
4	0.039822	0.0003811	0.001438

TABLE 5. Errors 2D example

hauto	$\|u^* - u_d\|_Q$	$\|y^* - y_h\|_Q$	$\lvert J(y^*, u^*) - J(y_h, u_h)\rvert$
7	0.3171	0.004792	0.016412
6	0.17107	0.0023017	0.0087468
5	0.050385	0.00054455	0.0020044

TABLE 6. Errors 2D example, adaptive solver

5.2. Sequential or iterative approach.

5.2.1. *The gradient method.* We use the solution operator G to eliminate y from the objective function. Our aim is now to minimize the functional $J(G(u), u)$ by the gradient method. For that, we have to evaluate the derivative

$$\begin{aligned}\left\langle \frac{d}{du} J(G(u), u), h \right\rangle &= \langle G(u) - y_d, Gh\rangle + \kappa \langle u, h\rangle \\ &= \langle G^* (G(u) - y_d), h\rangle + \kappa \langle u, h\rangle .\end{aligned}$$

where $h \in L^2(Q)$ is a directional vector. A direction of descent is given by

$$v = G^* (G(u) - y_d) + \kappa u.$$

Here $p := G^* (G(u) - y_d)) = G^*(y - y_d)$ is again the adjoint state given by the first equation in (7). The algorithm reads now

Algorithm 5.1. *(Gradient Method in function space)*

Choose $\epsilon > 0$. Choose u_{old} arbitrarily.
Initialize y_{old} by solving $y_{old} = G(u_{old})$.
while $v > \epsilon$
  solve the adjoint equation $p = G^* \left(G(u_{old}) - y_d\right)$
  set $v = p + \kappa(u_{old} - u_d)$
  for $k = 1, 2, ...$
    $u_{new} = u_{old} + \sigma v$
    solve the state equation $y_{new} = G(u_{new})$
    if $J(y_{new}, u_{new}) < J(y_{old}, u_{old})$
      break
    end
    set $\sigma = \sigma/2$
  end
  set $u_{old} = u_{new}$, $y_{old} = y_{new}$.
end

Remark 5.2. We point out that the gradient method is used as an "easy-to-implement" iterative method to illustrate COMSOL MULTIPHYSICS' capability of solving optimal control problems iteratively, i.e. advoiding the coupling of forward and backward PDEs, and thus also saving memory capacity. In realworld applications more sophisticated iterative methods with more elaborate line search strategies should be used.

To realize this algorithm in COMSOL MULTIPHYSICS, we define the optimality conditions by a system of PDEs like in the one-shot-approach instead of defining two separate PDEs. As a further difference to the one-shot-approach, we cannot replace u by $u = \frac{1}{\kappa}p - u_d$ and we explicitly need an additional helper variable u_{old}.

Although we have a system of PDEs, we have to solve the state and adjoint equation separately. Solving a single PDE will be done by using the `femtime` function for selected solution parts, i.e. for the adjoint p. The advantage of this approach is that all parts of the given PDE-system live on different meshes and time intervals, but the interpolation work will be performed automatically in COMSOL MULTIPHYSICS. The coupling $u_{old} = u_{new}$ will be realized by solving a trivial PDE. In fact, we implement the following PDE-system in COMSOL MULTIPHYSICS, here demonstrated for Example 1, and written in general form:

$$\begin{aligned} -\nabla \cdot \nabla y &= u_{new} - y_t && \text{in } Q \\ \vec{n} \cdot \nabla y &= 0 && \text{on } \Sigma \\ y(0) &= 0 && \text{in } \Omega \end{aligned} \qquad \begin{aligned} -\nabla \cdot \nabla p &= y - y_d + p_t && \text{in } Q \\ \vec{n} \cdot \nabla p &= 0 && \text{on } \Sigma \\ p(0) &= 0 && \text{in } \Omega \end{aligned}$$

$$\begin{aligned} u_{old} - \rho(p + \nu(u_{old} - u_d)) - u_{new} &= 0 \text{ in } Q \\ u_{old} - \rho(p + \kappa(u_{old} - u_d)) - u_{new} &= 0 \text{ on } \Sigma \cup \{0\} \times \Omega \cup \{T\} \times \Omega \\ u_{old} - u_{new} &= 0 \text{ in } Q \\ u_{old} - u_{new} &= 0 \text{ on } \Sigma \cup \{0\} \times \Omega \cup \{T\} \times \Omega \end{aligned}$$

To test this method, we will apply it to Examples 3.1.2–5.1.3.

5.2.2. *Solving the 1D example with distributed control by the iterative method.* In the following listings we present the COMSOL MULTIPHYSICS-code that solves Example 1 by the sequential method. In the code, we drop the subscript in u_{new} and write simply u. Further, we need only one state variable y. The setting $y_{old} = y_{new}$ is redundant because we save the value of $J(y_{old}, u_{old})$ instead. In Listing 7 we define the geometry as a solid-object (line 1).

```
%geometry and mesh:
(1)   fem.geom = solid1([0 pi]);
      fem.mesh = meshinit(fem,'hmax',2^(-3));
      fem.sdim = {'x'};
% unknowns and element types:
      fem.form = 'general';
```

```
      fem.dim = {'y' 'p' 'u' 'uold'};
      fem.shape = [1 1 1 1];
% parameters:
      rho=1;
      fem.const = {'rho' num2str(rho) 'nu' '1.e-2'};
```

LISTING 6.

The definition of the functions y_d, u_d and g as inline functions is the same as in Listing 2, lines 8–17, so we do not show here the lines 9–18 of our code.

The definition of the PDE-system can be done quite similar to the one-shot-approach. The general form of a time dependent PDE in COMSOL MULTIPHYSICS reads

$$a\frac{d}{dt}y + \nabla \cdot \Gamma = F \quad \text{in } \Omega.$$

The coefficient a will be stored in the structure `fem.equ.da`, see line 20. The second negative entry marks the backward-in-time adjoint equation, the third and fourth entries are zero, because the equations for u and u_{old} are non-differential. For a time-dependent PDE we have to set the initial values y_0 p_T, etc., this will be done in line 23. The coefficients `fem.equ.f` and `fem.bnd.r` are redefined before each call to `femtime`, depending on the variable to solve for, followed by a `meshextend` call. In lines 22 and 25 we initialize them by zeros. The reason for this "strange" programming is not easily seen: We define our problem as a system of PDEs, which results in an ill-defined problem because of the different time directions in state and adjoint. To overcome this ill-definedness, we solve it for every solution part separately by using e.g. the parameter `'solcomp',{'p'}`. See the call of `femtime` in lines 30, 51, 55, and 73. But, in this case `femtime` solves not only the PDE for p but all equations wherever p appears. Here, instead of solving the adjoint equation

$$\begin{aligned} -p_t - \nabla\cdot\nabla p &= y - y_d && \text{in } Q \\ \vec{n}\cdot\nabla p &= 0 && \text{on } \Sigma \\ p(0) &= 0 && \text{in } \Omega \end{aligned}$$

it tries to solve

$$\begin{aligned} -p_t - \nabla\cdot\nabla p &= y - y_d && \text{in } Q \\ \vec{n}\cdot\nabla p &= 0 && \text{on } \Sigma \\ p(0) &= 0 && \text{in } \Omega \\ u_{old} - rho(p + \nu(u_{old} - u_d)) - u_{new} &= 0 && \text{in } Q \\ u_{old} - \rho(p + \kappa(u_{old} - u_d)) - u_{new} &= 0 && \text{on } \Sigma \cup \{0\}\times\Omega \cup \{T\}\times\Omega \end{aligned}$$

which is a completely different problem. Therefore we have to redefine the problem each time we call `femtime`. In Listing 8 we give some comments in the relevant lines.

Notice, that the coupling between u and u_{old} is done by solving a pseudo-PDE with Dirichlet conditons on all bounds see lines 20–25. We have to define

```
fem.equ.f = { {0;0;'uold-rho*(p+nu*(uold-ud(x,t)))-u';0} }.
```

Since p runs backwards in time and also u and $uold$, we have to flip the time vektor for use in the state equation. This will be done in line 58 by the function `fliplr`. The flipped `tlist` is used during the solution for u and u_{old} in order to force an evaluation ot htese variables at the given time steps, cf. lines 51 and 73.. The listing shows that we have translated Algorithm 5.1 more or less directly into a COMSOL MULTIPHYSICS script.

```
% coefficients + rhd side:
      fem.globalexpr = {'y0' '0' 'J' '(y-yd(x,t))^2/2+nu/2*(u-ud(x,t))^2'...
                        'Ja' '(ya(x,t)-yd(x,t))^2/2+nu/2*(ua(x,t)-ud(x,t))^2'};
(20)  fem.equ.da = {{1 -1 0 0}};
(21)  fem.equ.ga = { { {'-yx'} {'-px'} {0} {0}} };
(22)  fem.equ.f = {{0 0 0 0}};
(23)  fem.equ.init = { {'y0' '0' 'ud(x,pi)' 'ud(x,pi)'} };
% boundary conditions:
(24)  fem.bnd.g = { {'g(t)' 0 0 0} };
(25)  fem.bnd.r = {{0 0 0 0}};
```

```
% auxilliary fem-struct:
      fem_sol = fem;
% gradient method to solve optimality system
% solve state equation for y
% redefine equations and boundary conditions
% y may only appear in state equation
(27)  fem.equ.f = { {'u';0;0;0} };
(28)  fem.bnd.r = { {0;0;0;0} };
(29)  fem.xmesh = meshextend(fem);
(30)  fem.sol = femtime(fem,'solcomp',{'y'},'tlist',...
                [0,pi],'tout','tsteps','maxstep', 0.01);

      J= postint(fem,'J','T',fem.sol.tlist);
      J= 1/2*(J(1:end-1)+J(2:end))*diff(fem.sol.tlist)';
      Jold = Inf;
      D = 1;   i = 0;
      while J<Jold
        Jold =J;
        i=i+1;
% solve adjoint equation for p
% redefine equations and boundary conditions
% p may only appear in adjoint equation
(38)    fem.equ.f = { {0;'y-yd(x,t)';0;0} };
        fem.bnd.r = { {0 0 0 0} };
(40)    fem.xmesh = meshextend(fem);
(41)    fem.sol = femtime(fem,'solcomp',{'p'},...
                  'outcomp',{'y','p','u','uold'},...
                  'u',fem.sol,'tlist',[pi,0],...
                  'tout','tsteps','maxstep', 0.01);
% save tlist
        tlist_rw = fem.sol.tlist;
        rho = 1;
        fem.const{2} = num2str(rho);
        j=0;
% determine step size
        while (J>=Jold & rho >=1.e-3)
          j=j+1;
% determine new control u
% redefine equations and boundary condtions
% u may only appear in third equation
          fem.equ.f = { {0;0;'uold-rho*(p+nu*(uold-ud(x,t)))-u';0} };
          fem.bnd.r = { {0;0;'uold-rho*(p+nu*(uold-ud(x,t)))-u';0} };
(50)      fem.xmesh = meshextend(fem);
(51)      fem.sol = femtime(fem,'solcomp',{'u'},...
                    'outcomp',{'y','p','u','uold'},...
                    'u',fem.sol,'tlist',tlist_rw,...
                    'tout','tsteps','tsteps','strict');
% solve state equation for y
% redefine equations and boundary conditions
% y may only appear in state equation
          fem.equ.f = { {'u';0;0;0} };
          fem.bnd.r = { {0;0;0;0} };
          fem.xmesh = meshextend(fem);
(55)      fem.sol = femtime(fem,'solcomp',{'y'},...
                    'outcomp',{'y','p','u','uold'},...
                    'u',fem.sol,'tlist',[0,pi],...
```

```
                     'tout','tsteps','maxstep', 0.01);
          J = postint(fem,'J','T',fem.sol.tlist);
          J =1/2*( J(2:end)+J(1:end-1))*diff(fem.sol.tlist)';

% flip tlist for use in third equation
          tlist_rw = fliplr(fem.sol.tlist);
          rho=rho/2;
(60)      fem.const{2} = num2str(rho);
        end
        if J<Jold
          fem_sol = fem;
          disp(['Successful step: J = ',num2str(J)]);
        else
          disp('Step failed')
        end
% change in control variables
        D = (postint(fem,'(uold-u)^2','T',fem.sol.tlist));
        D = 1/2*(D(1:end-1)+D(2:end))*diff(fem.sol.tlist)';

% save value of control u in uold
% redefine equations and boundary condtions
% uold may only appear in fourth equation
(70)    fem.equ.f = { {0;0;0;'u-uold'} };
        fem.bnd.r = { {0;0;0;'u-uold'} };
        fem.xmesh = meshextend(fem);
(73)    fem.sol = femtime(fem,'solcomp',{'uold'},...
                  'outcomp',{'y','p','u','uold'},...
                  'u',fem.sol,'tlist',tlist_rw,...
                  'tout','tsteps','tsteps','strict');
      end
      fem = fem_sol;
      clear fem_sol;
```

LISTING 7.

The postprocessing can be done similarly to the other examples. The posteval function in line 77 extracts a structure u from fem that contains only the solutions data. See Listing 9 for an example.

```
% post processing
(77)  u=posteval(fem,'u','solnum','all');
      subplot(222);
      surf(u.p(1:2:end),fem.sol.tlist(1:2:end),u.d(1:2:end,1:2:end))
```

LISTING 8.

Results. We solved Example 5.1.1 for a set of values h_{max}. Table shows the errors for u, y measured in the $L^2(Q)$-norm and for J, and the number of iterations needed for solving the problem. Obviously, the number of iterations depends on h_{max}.

5.2.3. *Solving Example 5.1.2 .* Secondly, we solved the boundary control problem 5.1.2 by the iterative method. Table 8 lists the errors and the values of J as well as the number of iterations. It is remarkable that the number of iterations is significantly smaller than in the example before.

5.2.4. *Solution to Example 5.1.3.* At last, we apply the iterative method to the 2D-Example 5.1.3.

The program code is analoguously to the one of the 1D example 5.1.1. The geometry is defined by

```
fem.geom = rect2(0, pi, 0, pi);
```

the Laplace operator is defined as

```
fem.equ.ga = { { {'-yx1' '-yx2'} {'-px1' '-px2'} {0 0} {0 0}} };
```

h_{max}	$\|u^* - u_h\|_Q$	$\|y^* - y_h\|_Q$	$J(y_h, u_h) - J(y^*, u^*)$	#it
2^0	3.3096	0.088064	0.022568	61
2^{-1}	0.63421	0.0056229	0.0029157	70
2^{-2}	0.10418	$4.1446 \cdot 10^{-4}$	$2.7439 \cdot 10^{-5}$	137
2^{-3}	0.035169	$1.6769 \cdot 10^{-4}$	$1.0575 \cdot 10^{-4}$	116
2^{-4}	0.030929	$1.2741 \cdot 10^{-4}$	$1.5912 \cdot 10^{-4}$	120
2^{-5}	0.018019	$5.6166 \cdot 10^{-5}$	$1.0325 \cdot 10^{-4}$	152
2^{-6}	0.0023372	$5.4141 \cdot 10^{-6}$	$7.0073 \cdot 10^{-6}$	316

TABLE 7. Distributed controled problem: Errors and values of J with respect to the parameter h_{max}.

h_{max}	$\|u^* - u_h\|_\Sigma$	$\|y^* - y_h\|_Q$	$\|J(y_h, u_h) - J(y^*, u^*)\|$	#it
2^0	0.060428	0.041127	$1.3985 \cdot 10^{-2}$	30
2^{-1}	0.0026638	0.0015048	$4.9358 \cdot 10^{-4}$	49
2^{-2}	$8.7139 \cdot 10^{-4}$	$1.4941 \cdot 10^{-4}$	$3.0021 \cdot 10^{-5}$	37
2^{-3}	0.0016595	$8.1605 \cdot 10^{-5}$	$6.4832 \cdot 10^{-5}$	29
2^{-4}	$3.0045 \cdot 10^{-4}$	$6.3345 \cdot 10^{-6}$	$1.7867 \cdot 10^{-5}$	54
2^{-5}	$3.2644 \cdot 10^{-4}$	$6.8206 \cdot 10^{-6}$	$7.4004 \cdot 10^{-6}$	52
2^{-6}	$5.2305 \cdot 10^{-4}$	$1.2691 \cdot 10^{-5}$	$9.7372 \cdot 10^{-6}$	44

TABLE 8. Boundary controled problem: Errors and values of J with respect to the parameter h_{max}.

where the independent variables are called `x1` and `x2`.

h_{max}	$\|u^* - u_h\|_Q$	$\|y^* - y_h\|_Q$	$\|J(y_h, u_h) - J(y^*, u^*)\|$	#it
2^0	39.290	0.19290	0.20213	271
2^{-1}	3.1704	0.025563	0.003234	81
2^{-2}	0.4500	0.0026441	0.002694	130
2^{-3}	0.0671	$2.6871 \cdot 10^{-4}$	$5.3364 \cdot 10^{-4}$	230
2^{-4}	comp. failed: memory overflow(?)			-

TABLE 9. Error to the 2D Example

6. CONCLUSIONS AND OUTLOOK

The finite element package COMSOL MULTIPHYSICS can be used for solving optimal control problems, provided the user offers the optimality system for the given problem. We studied two different approaches to solve systems of time dependent PDEs.

The one-shot-approach provides a very easily to implementable method to solve optimal control problems. Of course, the achieved results are not comparable with ones computed by specialized software. This has one main reason: In the one-shot-approach we used an implementation of the time dependent PDE which do not include the time as a "special" dimension. The problem becomes a singular elliptic problem. In such cases, stabilized methods are in use. Figure 4 shows one effect of this lack, the error flow through the space-time domain.

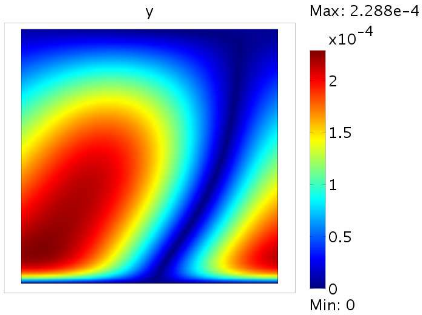


FIGURE 4. "Error flow" through the space-time domain. In a "correct" implementation, the error shold be symmetrically distributed.

The iterative method described in Algorithm 5.1 provides a mathematically correct formulation of the optimality conditions. Our implementation uses the capability of COMSOL MULTIPHYSICS to solve coupled systems of PDEs for each unknown separately. For a correct implementation of these methods, a deeper knowledge of COMSOL MULTIPHYSICS is necessary, e.g to avoid the "hidden" coupling of adjoint and state equation.

Altogether, COMSOL MULTIPHYSICS provides the possibility for a fast implementation of optimal control solvers. It does not redundantize specialized solvers for time-dependent optimal control problems.

A possible next step is to extend the methods described here to constrained optimal control problems. There, COMSOL MULTIPHYSICS's ability to solve e.g. barrier problems by symbolic differentiation should offer some chances. This will be the topic of the upcoming second part.

REFERENCES

[1] J. L. Lions. *Optimal Control of Systems Governed by Partial Differential Equations*. Springer-Verlag, Berlin, 1971.

[2] D. Meidner and B. Vexler. A priori error estimates for space-time finite element discretization of parabolic optimal control problems Part I: Problems without control constraints. submitted, 2007.

[3] U. Prüfert and F. Tröltzsch. An interior point method for a parabolic optimal control problem with regularized pointwise state constraints. *ZAMM*, 87(8–9):564–589, 2007.

[4] T. Slawig. PDE-constrained control using Femlab - Control of the Navier-Stokes equations. *Numerical Algorithms*, 42:107–126, 2006.

[5] F. Tröltzsch. *Optimale Steuerung partieller Differentialgleichungen. Theorie, Verfahren und Anwendungen*. Vieweg, Wiesbaden, 2005.

[6] J. Wloka. *Partielle Differentialgleichungen*. Teubner-Verlag, Leipzig, 1982.