"PDE optimization of the heat equation with time-varying boundary conditions"

December 4, 2017; rev. December 6, 2017

Simon Pirkelmann

1 Setting

Consider the following equation

$$y_t - \alpha \Delta y = 0 \text{ on } Q := \Omega \times [0, T] \tag{1}$$

where $y : Q \to \mathbb{R}$ is the temperature, $\alpha \in \mathbb{R}$ is the diffusion coefficient. We use the shorthand $y_t = \frac{\partial y}{\partial t}$ to denote the time derivative.

Let Ω be the domain. The boundary is partitioned in an boundary Γ_{out} where some outside temperature is prescribed and a control boundary Γ_c. On the control part of the boundary heating/cooling is applied which is described by a Dirichlet condition

$$y = u \text{ on } \Gamma_c. \tag{2}$$

On the other part we have

$$y = y_{out} \text{ on } \Gamma_{out} \tag{3}$$

where $\frac{\partial y}{\partial n}$ is the derivative of y in normal direction. We approximate the Dirichlet boundary conditions by using the following Robin type boundary condition instead

$$-\beta \frac{\partial y}{\partial n} = \gamma(y - z) \text{ on } \Gamma. \tag{4}$$

By choosing $\gamma := 10^3$, $\beta := 1$ and $z := \begin{cases} y_{out} & \text{on } \Gamma_{out} \\ u & \text{on } \Gamma_c \end{cases}$ we can approximate the Dirichlet boundary conditions. This will also allow us to extend the setting more easily in the future.

2 Numerical simulation of the heat equation

We simulate the equation using the finite element method.

2.1 Weak form

For the weak form we first discretize in time by picking a sampling rate $\Delta t > 0$. We define $y_k := y(\cdot, t_0 + \Delta t k)$, $y_{out,k} := y_{out}(\cdot, t_0 + \Delta t k)$, $u_k := u(t_0 + \Delta t k)$, $z_k := z(t_0 + \Delta t k)$ for $k \in \{0, 1, \dots, N\}$.
The initial value y_0 is given. To compute the next state y_{k+1} for each $k \in \{0, 1, \dots, N-1\}$ we replace y_t in equation (1) by $\frac{y_{k+1} - y_k}{\Delta t}$ and y by y_{k+1} using backward Euler. Multiplying with a test function v and integrating over the domain Ω yields

$$\int_\Omega \frac{y_{k+1} - y_k}{\Delta t} v \, dx - \alpha \int_\Omega \Delta y_{k+1} v \, dx = 0 \text{ for } k \in \{0, 1, \dots, N-1\}. \tag{5}$$

Using partial integration on the second integral we get

$$\int_\Omega \frac{y_{k+1}-y_k}{\Delta t} v \, dx - \alpha \int_\Gamma \frac{\partial y_{k+1}}{\partial n} v \, ds + \alpha \int_\Omega \nabla y_{k+1} \cdot \nabla v \, dx = 0 \tag{6}$$

Substituting the boundary condition (4) we obtain

$$\int_\Omega \frac{y_{k+1}-y_k}{\Delta t} v \, dx + \alpha \int_\Gamma \frac{\gamma}{\beta}(y_{k+1}-z_k) v \, ds + \alpha \int_\Omega \nabla y_{k+1} \cdot \nabla v \, dx = 0 \tag{7}$$

TODO: Strictly speaking, for backward Euler we may actually have to use z_{k+1} here instead. Think about this!
Substituting the definition of z_k on the different parts Γ_{out} and Γ_c of the boundary we get

$$\int_\Omega \frac{y_{k+1}-y_k}{\Delta t} v \, dx + \alpha \int_{\Gamma_c} \frac{\gamma}{\beta}(y_{k+1}-u_k) v \, ds + \alpha \int_{\Gamma_{out}} \frac{\gamma}{\beta}(y_{k+1}-y_{out,k}) v \, ds + \alpha \int_\Omega \nabla y_{k+1} \cdot \nabla v \, dx = 0 \tag{8}$$

2.2 Implementation in FEniCS

The above variational equation is implemented in FEniCS to obtain the system matrices for use in the optimization.

Reordering the terms in (8) by terms depending on the state y_{k+1} and all external terms we get:

$$\underbrace{\int_\Omega \frac{y_{k+1}}{\Delta t} v + \alpha \nabla y_{k+1} \cdot \nabla v \, dx + \alpha \int_\Gamma \frac{\gamma}{\beta} y_{k+1} v \, ds}_{a} = \underbrace{\int_\Omega \frac{y_k}{\Delta t} v \, dx}_{f_y} + \underbrace{\alpha \int_{\Gamma_c} \frac{\gamma}{\beta} u_k v \, ds}_{f_u} + \underbrace{\alpha \int_{\Gamma_{out}} \frac{\gamma}{\beta} y_{out,k} v \, ds}_{f_{y,out}} \tag{9}$$

This is a linear system of equations which we can explicitely assemble in FEniCS using the `assemble()` function. We obtain a system of the form

$$A y_{k+1,h} = B_y y_{k,h} + b_u u_k + b_{y,out} y_{out,k} \tag{10}$$

making use of the fact that u_k and $y_{out,k}$ are constant on the boundary. The system matrices and vectors are generated in FEniCS and then saved to a file for subsequent use in MATLAB.

```
# Prepare a mesh
mesh = UnitIntervalMesh(100)

# specify subdomains for the boundary
left = CompiledSubDomain("near(x[0], 0.)")
right = CompiledSubDomain("near(x[0], 1.)")
boundary_parts = MeshFunction("size_t", mesh, mesh.topology().dim() - 1
                              )
left.mark(boundary_parts, 0)     # boundary part where control is
                                   applied
right.mark(boundary_parts, 1)    # boundary part for outside temperature
ds = Measure("ds", subdomain_data=boundary_parts)

# Define function space
```

```
U = FunctionSpace(mesh, "Lagrange", 1)

# Define test and trial functions
v = TestFunction(U)
y = TrialFunction(U)
y0 = TrialFunction(U)

# Define constants
alpha = Constant(0.75)
beta = Constant(1.0)
gamma = Constant(1.0e3)
u = Constant(1.0)
y_out = Constant(1.0)

# Define variational formulation
a = (y / k * v + alpha * inner(grad(y), grad(v))) * dx + alpha * gamma/
                                        beta * y * v * ds
f_y = y0 / k * v * dx
f_u = alpha * gamma/beta * u * v * ds(0)
f_y_out = alpha * gamma/beta * y_out * v * ds(1)

# Assemble system matrices
A = assemble(a)
B_y = assemble(f_y)
b_u = assemble(f_u)
b_y_out = assemble(f_y_out)

# output matrices for use in matlab optimization
scipy.io.savemat('sys.mat', {'A': A.array(), 'B_y': B_y.array(), 'b_u':
                                        b_u.array(), 'b_y_out': b_y_out.
                                       array(), 'u': us, 'y_out': y_outs})
```

3 Optimal Control Problem

Our goal is to solve the following problem:

$$\begin{aligned}\min_{y,u} J(y,u) =& \frac{\varepsilon}{2}\int_\Omega (y(x,T)-y_\Omega(T,x))^2\,dx + \frac{\varepsilon}{2}\int_0^T\int_\Omega (y(x,t)-y_\Omega(t,x))^2\,dx\,dt \\ &+ \frac{1}{2}\int_0^T\int_{\Gamma_c}(u(x,t))^2\,ds\,dt \\ \text{s.t.} &(1),(4) \\ &\underline{u}(x,t) \le u(x,t) \le \overline{u}(x,t) \\ &\underline{y}(x,t) \le y(x,t) \le \overline{y}(x,t) \text{ on } \Omega_y\end{aligned}$$

where $\Omega_y \subset \Omega$.
The problem is solved by the First-Discretize-Then-Optimize-Approach. We replace y by its finite element approximations $y_{h,k}$ for each time point $t = k\Delta t, k \in \{0,\dots,N\}$. Let $y_h = (y_{h,0},\dots,y_{h,N})$ and $u = (u_0,\dots,u_{N-1})$. We end up with a finite dimensional

optimal control problem which reads:

$$\begin{aligned}
&\min_{y_h,u} J(y_h,u) = \sum_{k=0}^{N-1} \frac{\varepsilon}{2}(y_{h,k}-y_{\Omega,k})^T Q(y_{h,k}-y_{\Omega,k}) + \frac{1}{2}(u_k-u_{ref,k})^T R(u_k-u_{ref,k}) \\
&+ \frac{\varepsilon}{2}(y_{h,N}-y_{\Omega,N})^T Q(y_{h,N}-y_{\Omega,N}) \\
\text{s.t. } & Ay_{k+1,h} = B_y y_{h,k} + b_u u_k + b_{y,out} y_{out,k} \text{ for } k \in \{0,\dots,N\} \\
& \underline{y}_{h,k,i} \le y_{h,k,i} \le \overline{y}_{h,k,i} \text{ for } k \in \{0,\dots,N\}, i \in \mathcal{I}_{\Omega_y} \\
& \underline{u}_k \le u_k \le \overline{u}_k \text{ for } k \in \{0,\dots,N-1\}
\end{aligned}$$

where $\mathcal{I}_{\Omega_y}$ is the set of all indices with finite element nodes within the set Ω_y. When using Lagrange finite elements the degrees of freedom of the FE approximation y_h of the state y correspond to the value of y_h at the finite element nodes. This means we can enforce the state constraint pointwise at the finite element nodes, simply by constraining the corresponding elements of y_h.

Introducing $z = (y_h, u)$ the equality constraint of the optimization problem can be written as a single linear equation

$$A_{eq} z = b_{eq} \tag{11}$$

where

$$A_{eq} = \underbrace{\begin{pmatrix} -B_y & A & & & & b_u & & \\ & -B_y & A & & & & b_u & \\ & & \ddots & & & & & \ddots \\ & & & -B_y & A & & & b_u \end{pmatrix}}_{\in \mathbb{R}^{Nn_y \times (N+1)n_y + (N-1)n_u}} \tag{12}$$

$$b_{eq} = \underbrace{\begin{pmatrix} b_{y,out} & & & \\ & b_{y,out} & & \\ & & \ddots & \\ & & & b_{y,out} \end{pmatrix}}_{\in \mathbb{R}^{Nn_y \times (N-1)}} y_{out} \in \mathbb{R}^{Nn_y} \tag{13}$$

where n_y and n_u denote the dimensions of $y_{h,k}$ and u_k, respectively. Note that this is a sparse linear system and should be treated accordingly (i.e. only allocate memory for nonzero matrix entries).

The optimization algorithm also needs to be provided with the gradient and (optionally) the hessian of the cost functional J. Those quantities are given by

$$\nabla J(z) = \begin{pmatrix} \varepsilon Q(y_{h,0} - y_{\Omega,0}) \\ \vdots \\ \varepsilon Q(y_{h,N} - y_{\Omega,N}) \\ R(u_0 - u_{ref,0}) \\ \vdots \\ R(u_{N-1} - u_{ref,N-1}) \end{pmatrix} \tag{14}$$

$$\nabla^2 J(z) = \begin{pmatrix} \varepsilon Q & & & & & \\ & \ddots & & & & \\ & & \varepsilon Q & & & \\ & & & R & & \\ & & & & \ddots & \\ & & & & & R \end{pmatrix}. \tag{15}$$

The hessian is also a sparse matrix.

For implementation in MATLAB this is all that is necessary. For implementation in Ipopt we additionally have to provide the jacobian of the constraints as well as the hessian of the Lagrange function (see https://www.coin-or.org/Ipopt/documentation/node22.html). For Ipopt we rewrite equation (11) to be of the form $g(z) = 0$, i.e.

$$g(z) = A_{eq} z - b_{eq} = 0. \tag{16}$$

Then the Jacobian of g is given by

$$\nabla g = A_{eq} \tag{17}$$

Because we have a linear system the second order derivatives of g vanish and thus the hessian of the Lagrangian coincides with the hessian of the cost functional.

NOTE: Ipopt provides options for indicating that the jacobian and hessian matrices are constant. These can be activated in our case, since we have a quadratic optimization problem and linear equality constraints. This may speed up the optimization.

4 Example

Consider the following example. Let Ω be the unit interval $[0, 1]$ and let $\Omega_y := [\frac{1}{4}, \frac{3}{4}]$ and let Γ_{out} be the left boundary (for $x = 0$) and Γ_c be the right boundary (for $x = 1$). The domains and subdomains are illustrated in Figure 1.

Figure 1: Illustration of the domain.

The mesh is discretized by $n_y = 100$ finite elements. The dimension of the control is $n_u = 1$. We pick a sampling rate $\Delta t = 10^{-2}$. At the boundary Γ_{out} we have the time-varying boundary value function

$$y_{out}(t) = \frac{1}{2} + \frac{1}{3}\sin(10t). \tag{18}$$

For discrete time points $t = k\Delta t$ this corresponds to

$$y_{out,k} = \frac{1}{2} + \frac{1}{3}\sin(\frac{k}{10}). \tag{19}$$

We choose the parameters of the PDE system

$$\begin{aligned}\alpha &= 1\\ \beta &= 1\\ \gamma &= 10^3\end{aligned}$$

and parameters for the optimal control problem

$$\begin{aligned}\varepsilon &= 10^{-3}\\ Q &= I_{n_y \times n_y}\\ R &= I_{n_u \times n_u} = 1\\ N &= 10\\ y_{\Omega,k} &= 0.5 \text{ for } k \in \{0,\dots,N\}\\ u_{ref,k} &= 0.5 \text{ for } k \in \{0,\dots,N-1\}.\end{aligned}$$

The constraints for control and state are given by

$$\begin{aligned}\underline{y}_{h,k,i} &= 0.35 \text{ for } k \in \{0,\dots,N\}, i \in \mathcal{I}_{\Omega_y}\\ \overline{y}_{h,k,i} &= 0.65 \text{ for } k \in \{0,\dots,N\}, i \in \mathcal{I}_{\Omega_y}\\ \underline{u}_k &= 0.25 \text{ for } k \in \{0,\dots,N-1\}\\ \overline{u}_k &= 0.75 \text{ for } k \in \{0,\dots,N-1\}.\end{aligned}$$

In the case that we use Langrange finite elements we have that $\mathcal{I}_{\Omega_y} = \{\frac{n_y}{4},\dots,\frac{3n_y}{4}\}$.

Reference