Optimal Control of the Poisson Equation with OpenFOAM

Victor Hernández-Santamaría * José Vicente Lorenzo Gómez †

June 19, 2018

In this tutorial, we show how to use the C++ library OpenFOAM (Open Field Operation and Manipulation) in order to solve control problems for partial differential equations (PDE).

OpenFOAM is a free open source toolbox that allows the user to code customized solvers for Continuum Mechanics, with special attention to the field of Computational Fluid Dynamics. One can either use one of the multiple solvers already programmed for steady-state, transient or multiphase problems in fluid mechanics, or code his own solver depending on particular needs.

In this work, we will solve the optimal control problem

$$\min_{u\in L^2(\Omega)} \mathcal{J}(u) = \min_{u\in L^2(\Omega)} \frac{1}{2}\int_\Omega (y-y_d)^2 \,\mathrm{d}\Omega + \frac{\beta}{2}\int_\Omega u^2 \mathrm{d}\Omega, \tag{1}$$

where u is the control variable, y the state variable and y_d a target function. The minimization problem is subject to the elliptic partial differential equation

$$\begin{cases} -\Delta y = f + u & \text{in } \Omega, \\ y = 0 & \text{on } \Gamma. \end{cases} \tag{2}$$

We use some classical gradient descent methods based on the adjoint methodology, namely the steepest descent and conjugate gradient methods. The corresponding adjoint system for (2) writes as,

$$\begin{cases} -\Delta \lambda = y - y_d & \text{in } \Omega, \\ \lambda = 0 & \text{on } \Gamma. \end{cases} \tag{3}$$

In the steepest descent gradient method, we compute the directional derivative of the cost function in (1),

$$\mathcal{D}_{\delta u}\mathcal{J}(u) = \int_\Omega (\lambda + \beta u)\,\delta u \mathrm{d}\Omega, \tag{4}$$

where λ is solution to the elliptic problem in (3), and use the gradient to update the control variable as

$$u^{(n+1)} = u^{(n)} - \gamma\left(\lambda^{(n)} + \beta u^{(n)}\right), \tag{5}$$

for some value of γ.

One of the main advantages of OpenFOAM is its friendly syntax to describe PDE, as can be seen in the lines 10 and 13 in the code below.

```
1  // Initialize L2 norm of control update
2  scalar pL2 = 10*tol;
3
4  // Get the volume of mesh cells
5  scalarField volField = mesh.V();
6
7  while (runTime.loop() && ::sqrt(pL2) > tol)
8  {
9      // Solve primal equation
10     solve(fvm::laplacian(k, y) + u);
11
12     // Solve ajoint equation
13     solve(fvm::laplacian(k, lambda) + y - yd);
14
15     // Update control
16     p = lambda + beta*u;
17     u = u - gamma*p;
```

*University of Deusto
†Universidad Autónoma de Madrid.

```
18
19      // Control update norm
20      pL2 = ::sqrt(gSum(volField*p.internalField()*p.internalField()));
21
22      // Compute cost function
23      J = 0.5*gSum(volField*(Foam::pow(y.internalField()-yd.internalField(),2) \
24        + beta*Foam::pow(u.internalField(),2)));
25
26      // Display information
27      Info << "Iteration " << runTime.timeName() << " - " \
28        << "Cost value " << J << " - " \
29        << "Control variation L2 norm " << ::sqrt( pL2 ) << endl;
30
31      // Save current iteration results
32      runTime.write();
33  }
```

steepestdescent.cpp

In order to improve the covergence to the optimal control, the conjugate gradient method is applied also to the problem. First, the state variable must be separated in two terms as

$$y = y_u + y_f, \tag{6}$$

where y_u solves the state equation with zero Dirichlet boundary conditions,

$$\begin{cases} -\Delta y_u = u & \text{in } \Omega, \\ y_u = 0 & \text{on } \Gamma, \end{cases} \tag{7}$$

and y_f is the control-free solution to the state equation,

$$\begin{cases} -\Delta y_f = f & \text{in } \Omega, \\ y_f = 0 & \text{on } \Gamma. \end{cases} \tag{8}$$

Now, using the above separation of the state variable, one part depending on the control and the other part on a known source term, the cost functional in (1) can be expressed as

$$\mathcal{J}(u) = \frac{1}{2}\left(y_u + y_f - y_d, y_u + y_f - y_d\right)_{L^2(\Omega)} + \frac{\beta}{2}\left(u, u\right)_{L^2(\Omega)}. \tag{9}$$

We define a linear operator $\Lambda : L^2(\Omega) \to L^2(\Omega)$ that takes a control u and returns the solution to problem (7)-(**??**), so that $y_u = \Lambda u$. We introduce as well its adjoint operator $\Lambda^* : L^2(\Omega) \to L^2(\Omega)$ that takes the source term in (3) and solves the Poisson equation, so that $\lambda = \Lambda^*(y - y_d)$.

The directional derivative of the functional (9) then reads as

$$\mathcal{D}_{\delta u}\mathcal{J}(u) = \left(\underbrace{\left(\Lambda^*\Lambda + \beta I\right)}_{A_{cg}} u - \underbrace{\Lambda^*\left(y_d - y_f\right)}_{b_{cg}}, \delta u \right)_{L^2(\Omega)}. \tag{10}$$

After having identified A_{cg} and b_{cg} we can use the conjugate gradient method to reach the optimal control faster.

```
// Compute b: right-hand side of Ax = b
solve(fvm::laplacian(k, y));
solve(fvm::laplacian(k, lambda) + yd - y);
volScalarField b = lambda;

// Compute A*f0
// Primal equation
solve(fvm::laplacian(k, y0) + u);
// Adjoint equation
solve(fvm::laplacian(k, lambda) + y0);

// Compute g0: initial gradient
volScalarField g = lambda + beta*u - b;
scalar gL2 = gSum( volField * g.internalField() * g.internalField() );
scalar gL2a = 0;

// Compute initial residual and its norm
volScalarField r = -g;
```

Algorithm 1 Optimal control with Conjugate Gradient Method

Require: y_D, $u^{(0)}$, β, y_d, tol

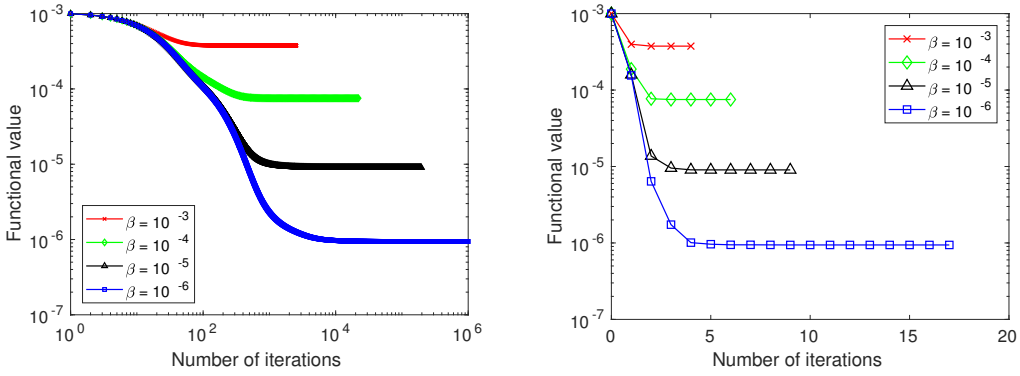
$n \leftarrow 0$
compute the control-free solution, y_f
$b \leftarrow \Lambda^* (y_d - y_f)$
$z \leftarrow \Lambda u$
$g \leftarrow \Lambda^* z + \beta u - b$
$h \leftarrow ||g||^2_{L^2([0,T])}$
$h_a \leftarrow h$
$r \leftarrow -g$
while $||r||_{L^2([0,T])} > tol$ **do**
    $z \leftarrow \Lambda r$
    $w \leftarrow \Lambda^* z + \beta r$
    $\alpha \leftarrow \frac{h}{(r,w)_{L^2([0,T])}}$
    $u \leftarrow u + \alpha r$
    $g \leftarrow g + \alpha w$
    $h_a \leftarrow h$
    $h \leftarrow ||g||^2_{L^2([0,T])}$
    $\gamma \leftarrow \frac{h}{h_a}$
    $r \leftarrow -g + \gamma r$
    $n \leftarrow n + 1$

```
scalar rL2 = gL2;

volScalarField w = g;

scalar alpha = 0;
scalar gamma = 0;

while (runTime.loop() && ::sqrt( rL2 ) > tol)
{
    r.dimensions().reset( u.dimensions() );

    // Compute w = A*r
    solve(fvm::laplacian(k, y0) + r);
    solve(fvm::laplacian(k, lambda) + y0);
    w = lambda + beta*r;

    // Update alpha
    alpha = gL2 / gSum( volField * r.internalField() * w.internalField() );

    // Update control
    u = u + alpha*r;

    // Update cost gradient and its norm
    g = g + alpha*w;
    gL2a = gL2;
    gL2 = gSum( volField * g.internalField() * g.internalField() );
    gamma = gL2/gL2a;

    // Update residual and its norm
    r.dimensions().reset( g.dimensions() );
    r = -g + gamma*r;
    rL2 = gSum( volField * r.internalField() * r.internalField() );

    solve(fvm::laplacian(k, y) + u);

    J = 0.5*gSum(volField*(Foam::pow(y.internalField()-yd.internalField(),2) \
     + beta*Foam::pow(u.internalField(),2)));

    Info << "Iteration no. " << runTime.timeName() << " - " \
      << "Cost value " << J << " - " \
      << "Residual " << ::sqrt( rL2 ) << endl;

    runTime.write();
}
```

conjugategradient.cpp

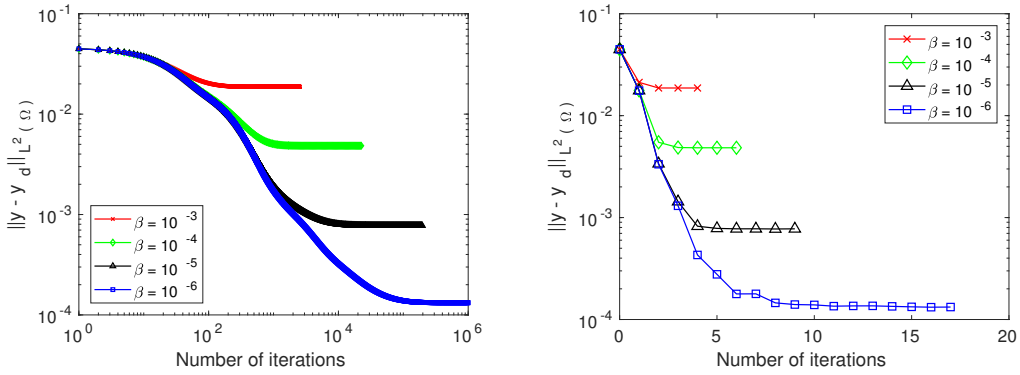
Both solvers, *laplaceAdjointFoam* for the steepest descent method with $\gamma = 10$ and *laplaceCGAdjointFoam* for the conjugate gradient method, have been tested in a square domain $[0,1]\times[0,1]$ with zero Dirichlet boundary conditions and $\beta = 10^{-3}, 10^{-4}, 10^{-5}, 10^{-6}$. The target function is $y_d = xy\sin(\pi x)\sin(\pi y)$.

The problem setup is called *case* in OpenFOAM and it is independent from the solver itself. A typical case in OpenFOAM has three folders: *0*, where the problem fields are stored along with their boundary conditions; *constant*, that includes the mesh data and the physical properties; and *system*, with parameters regarding the numerical solution and problem ouput such as tolerances, linear solvers or write interval. In this example, the case folder is called *laplaceAdjointFoamCase*.



(a) Steepest descent method. (b) Conjugate gradient method.

Figure 1: Functional value against the number of iterations of the gradient method.



(a) Steepest descent method. (b) Conjugate gradient method.

Figure 2: $||y - y_d||_{L^2(\Omega)}$ against the number of iterations of the gradient method.

References

[1] *The OpenFOAM Foundation*, openfoam.org.

[2] F. Tröltzsch. *Optimal control of partial differential equations: theory, methods, and applications*. American Mathematical Soc., 2010.