

# Optimal Control of the Poisson Equation with OpenFOAM

Victor Hernández-Santamaría  
University of Deusto

José Vicente Lorenzo Gómez  
Universidad Autónoma de Madrid

Enrique Zuazua  
University of Deusto

May 2, 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

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

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

$$-\Delta y = f + u \quad \text{in } \Omega, \quad (2)$$

$$y = 0 \quad \text{on } \Gamma. \quad (3)$$

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,

$$-\Delta \lambda = y - y_d \quad \text{in } \Omega, \quad (4)$$

$$\lambda = 0 \quad \text{on } \Gamma. \quad (5)$$

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

$$J'(u) \delta u = \int_{\Omega} (\lambda + \beta u) \delta u d\Omega, \quad (6)$$

where  $\lambda$  is solution to the elliptic problem in (4), and use the gradient to update the control variable as

$$u^{(n+1)} = u^{(n)} - \gamma (\lambda^{(n)} + \beta u^{(n)}), \quad (7)$$

for some value of  $\gamma$ .

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;
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 coverage 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

$$y = y_u + y_f, \quad (8)$$

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

$$-\Delta y_u = u \quad \text{in } \Omega, \quad (9)$$

$$y_u = 0 \quad \text{on } \Gamma, \quad (10)$$

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

$$-\Delta y_f = f \quad \text{in } \Omega, \quad (11)$$

$$y_f = 0 \quad \text{on } \Gamma. \quad (12)$$

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

$$J(u) = \frac{1}{2} (y_u + y_f - y_d, y_u + y_f - y_d)_{L^2(\Omega)} + \frac{\beta}{2} (u, u)_{L^2(\Omega)}. \quad (13)$$

We define a linear operator  $\Lambda : L^2(\Omega) \rightarrow L^2(\Omega)$  that takes a control  $u$  and returns the solution to problem (9)-(10), so that  $y_u = \Lambda u$ . We introduce as well its adjoint operator  $\Lambda^* : L^2(\Omega) \rightarrow L^2(\Omega)$  that takes the source term in (4) and solves the Poisson equation, so that  $\lambda = \Lambda^*(y - y_d)$ .

The directional derivative of the functional (13) then reads as

$$J'(u) \delta u = \left( \underbrace{(\Lambda^* \Lambda + \beta I) u}_{A_{cg}} - \underbrace{\Lambda^*(y_d - y_f)}_{b_{cg}}, \delta u \right)_{L^2(\Omega)}. \quad (14)$$

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

```

1 // Compute b: right-hand side of Ax = b
2 solve(fvm::laplacian(k, y));
3 solve(fvm::laplacian(k, lambda) + yd - y);
4 volScalarField b = lambda;
5
6 // Compute A*f0
7 // Primal equation
8 solve(fvm::laplacian(k, y0) + u);
9 // Adjoint equation
10 solve(fvm::laplacian(k, lambda) + y0);
11
12 // Compute g0: initial gradient
13 volScalarField g = lambda + beta*u - b;
14 scalar gL2 = gSum( volField * g.internalField() * g.internalField() );
15 scalar gL2a = 0;
16

```

---

**Algorithm 1** Optimal control with Conjugate Gradient Method

---

**Require:**  $y_D, u^{(0)}, \beta, y_d, tol$

```
1:  $n \leftarrow 0$ 
2: compute the control-free solution,  $y_f$ 
3:  $\bar{b} \leftarrow \Lambda^*(y_d - y_f)$ 
4:  $z \leftarrow \Lambda u$ 
5:  $g \leftarrow \Lambda^* z + \beta u - b$ 
6:  $h \leftarrow \|g\|_{L^2([0,T])}^2$ 
7:  $h_a \leftarrow h$ 
8:  $r \leftarrow -g$ 
9: while  $\|r\|_{L^2([0,T])} > tol$  do
10:    $z \leftarrow \Lambda r$ 
11:    $w \leftarrow \Lambda^* z + \beta r$ 
12:    $\alpha \leftarrow \frac{h}{(r,w)_{L^2([0,T])}}$ 
13:    $u \leftarrow u + \alpha r$ 
14:    $g \leftarrow g + \alpha w$ 
15:    $h_a \leftarrow h$ 
16:    $h \leftarrow \|g\|_{L^2([0,T])}^2$ 
17:    $\gamma \leftarrow \frac{h}{h_a}$ 
18:    $r \leftarrow -g + \gamma r$ 
19:    $n \leftarrow n + 1$ 
```

---

```
17 // Compute initial residual and its norm
18 volScalarField r = -g;
19 scalar rL2 = gL2;
20
21 volScalarField w = g;
22
23 scalar alpha = 0;
24 scalar gamma = 0;
25
26 while (runTime.loop() && ::sqrt( rL2 ) > tol)
27 {
28   r.dimensions().reset( u.dimensions() );
29
30   // Compute w = A*r
31   solve(fvm::laplacian(k, y0) + r);
32   solve(fvm::laplacian(k, lambda) + y0);
33   w = lambda + beta*r;
34
35   // Update alpha
36   alpha = gL2 / gSum( volField * r.internalField() * w.internalField() );
37
38   // Update control
39   u = u + alpha*r;
40
41   // Update cost gradient and its norm
42   g = g + alpha*w;
43   gL2a = gL2;
44   gL2 = gSum( volField * g.internalField() * g.internalField() );
45   gamma = gL2/gL2a;
46
47   // Update residual and its norm
48   r.dimensions().reset( g.dimensions() );
49   r = -g + gamma*r;
50   rL2 = gSum( volField * r.internalField() * r.internalField() );
51
52   solve(fvm::laplacian(k, y) + u);
53
54   J = 0.5*gSum(volField*(Foam::pow(y.internalField()-y_d.internalField(),2) \
55     + beta*Foam::pow(u.internalField(),2)));
56
57   Info << "Iteration no. " << runTime.timeName() << " - " \
58     << "Cost value " << J << " - " \
59     << "Residual " << ::sqrt( rL2 ) << endl;
60
61   runTime.write();
62 }
```

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 output such as tolerances, linear solvers or write interval. In this example, the case folder is called *laplaceAdjointFoamCase*.

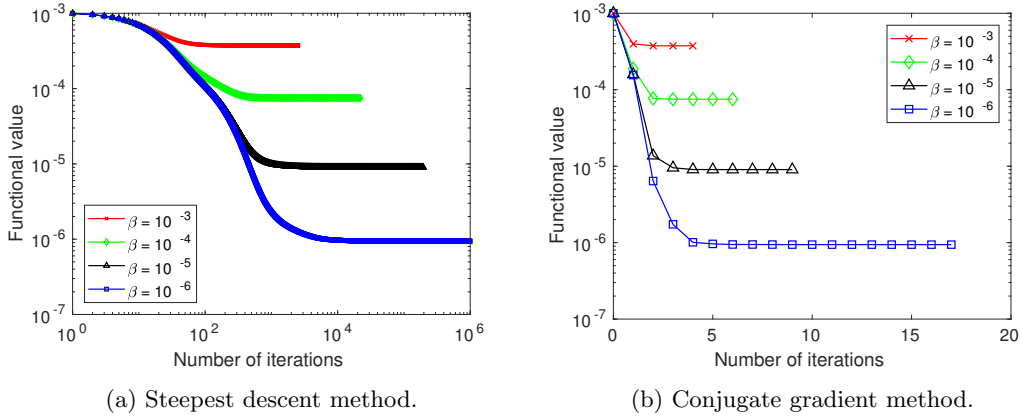


Figure 1: Functional value against the number of iterations of the gradient method.

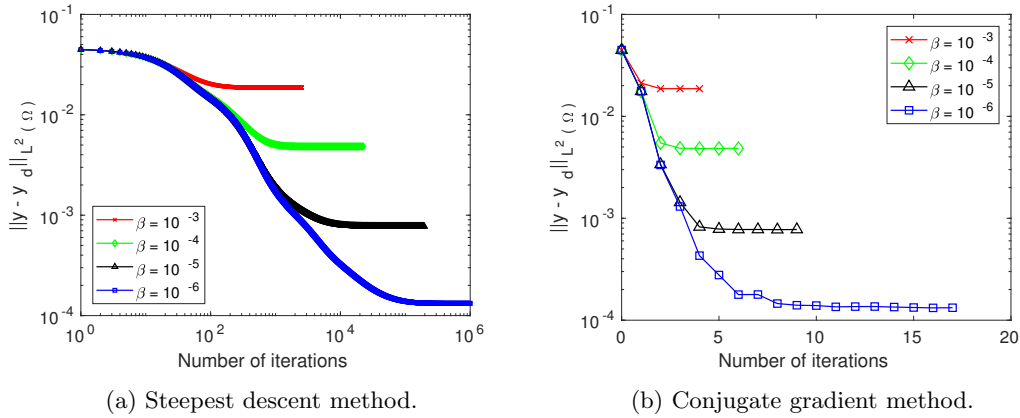


Figure 2:  $\|y - y_d\|_{L^2(\Omega)}$  against the number of iterations of the gradient method.

## References

- [1] *The OpenFOAM Foundation*, openfoam.org.
- [2] F. Tröltzsch. *Optimal control of partial differential equations: theory, methods, and applications*. American Mathematical Soc., 2010.