Algorithmic Differentiation for shape derivatives with PDE-constraints

Jørgen S. Dokken, Simon W. Funke and Stephan Schmidt

Automatically derive shape derivatives of FEniCS models

Highlights

- We present an extension to *dolfin-adjoint* which computes shape derivatives.
- The extension computes shape derivatives of time-dependent, linear and non-linear FEniCS models with minimal code changes.

Mathematical formulation

For shape optimization problems, the design variable is the computational domain Ω of the functional J and state equations. For efficient optimization, one needs to introduce the notion of a shape derivative

$$\mathrm{d}J(\Omega)[s] := \lim_{\epsilon \to 0^+} \frac{J(\Omega(\epsilon)[s]) - J(\Omega)}{\epsilon},$$

where a change in the domain Ω is described with a displacement function s

$$\Omega(\epsilon)[s] := \{x + \epsilon s(x) : x \in \Omega\}.$$

If the functional is a combination of volume and surface integrals, shape calculus can be employed to find an integral representation of the shape derivative.

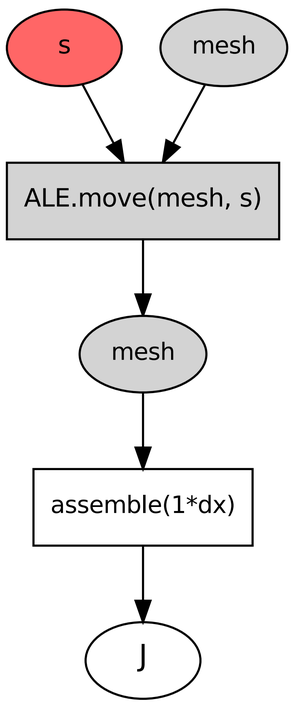
How it works

While the *FEniCS* model is executed, *dolfin-adjoint* [1] records operations in a computational graph. Our extension records all *Mesh* initializations and *ALE.move* calls. Using this graph, *dolfin-adjoint* uses *FEMORPH* [2], which employs shape calculus to compute shape derivatives of the recorded output.

```
from dolfin import *
from dolfin_adjoint import *
mesh = Mesh(UnitSquareMesh(100, 100))
V = VectorFunctionSpace(mesh, "CG", 1)
s = Function(V)
ALE.move(mesh, s)
J = assemble(1*dx(mesh))
c = Control(s)
Jhat = ReducedFunctional(J, c)
Jhat.derivative()
```

△ **Code**: Complete code for computing a shape derivative. The control is the mesh deformation vector s.

Figure: ▷
Visualization of the computation graph after executing the above code. The control variable is shown in red.



Future work

- Support strong enforcement of Dirichlet boundary conditions at control boundaries.
- Add re-meshing with *gmsh* as an alternative to *ALE.move* to update the computational domain.
- Add support for time-dependent domains and tube derivatives.

References

[1] S. W. Funke, S. Mitusch. A new algorithmic differentiation tool (not only) for FEniCS, *Poster at FEniCS 17*, 2017.

[2] S. Schmidt. Weak and Strong Form Shape Hessians and their Automatic Generation, *Journal of Scientific Computing*, in press.

Example: 3D linear elasticity

Consider the linear elasticity equation

$$\nabla \cdot \sigma = f \quad \text{in } \Omega, \quad \sigma \cdot n = g \quad \text{on } \Gamma, \quad u = 0 \quad \text{on } \partial\Omega \setminus \Gamma,$$

where $\sigma = 2\mu\epsilon(u) + \lambda tr(\epsilon(u))I$, $\epsilon(u) = \frac{1}{2}(\nabla u + (\nabla u)^T)$, u is the unknown displacement and Ω is a 3D beam. Here f are the body forces, g the surface forces. The goal is to minimize the structural compliance:

$$\min_{u,\Omega} J(u) = \min_{u,\Omega} \int_\Omega \sigma(u) : \epsilon(u) \, \mathrm{d}x.$$

```
ALE.move(mesh, s)
a = inner(sigma(u), grad(v))*dx
L = inner(f, v)*dx  + inner(g, v)*ds
solve(a==L, u_, bcs)
J = assemble(inner(sigma(u_),sym(grad(u_)))*dx)
Jhat = ReducedFunctional(J, Control(s))
s_optimal = minimize(Jhat)
```

△ **Code**: Simplified FEniCS code for the linear elasticity problem. The complete optimization code is $\simeq$ 120 lines.

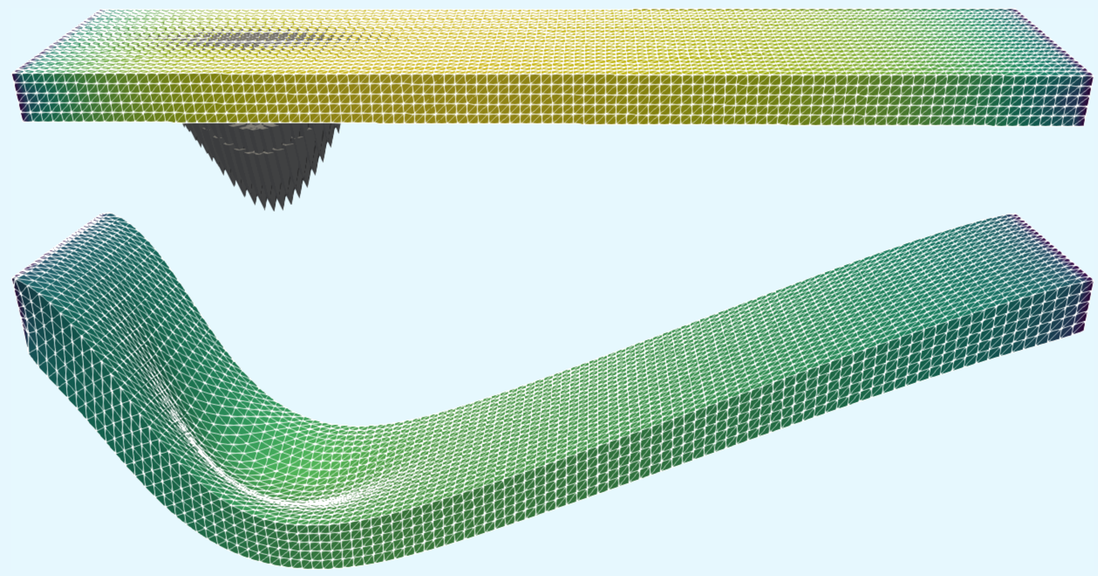


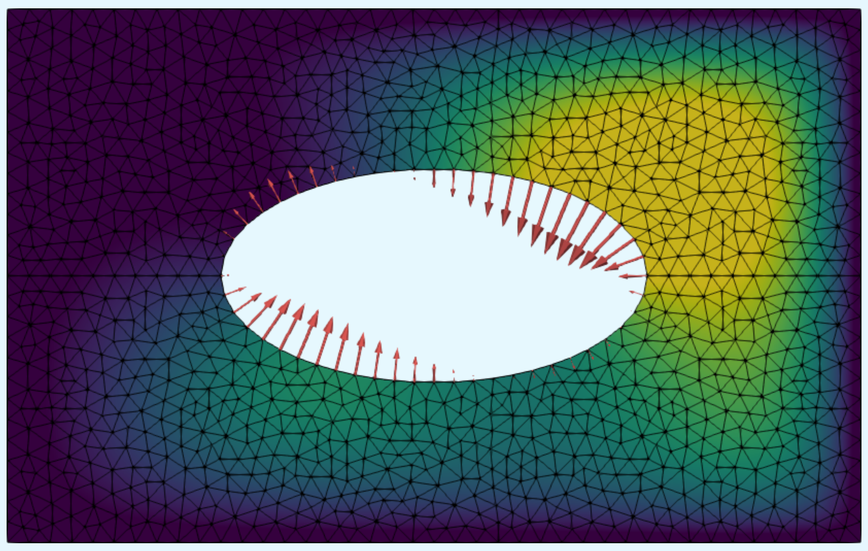
Figure: Initial beam and optimized beam. The vectors on the initial beam illustrates the traction $\sigma \cdot n$ on the top of the beam. The colors indicate the magnitude of the displacement on a logarithmic scale. The optimization takes 13 iterations, where $J_{Initial} = 1162$, $J_{optimized} = 184$.

Example: Heat equation

Consider the heat equation and corresponding functional

$$\frac{\partial u}{\partial t} - \Delta u = f, \quad \text{in } \Omega \times (0, T), \quad J(u) = \int_0^T \int_\Omega u^2 \, dx \, dt.$$

In these equations, Ω is the domain below. We impose a homogeneous Neumann condition on Γ, the boundary of the ellipsoid, and a Dirichlet condition on the remaining boundary.



△ **Figure**: The solution $u(t)$ at $\frac{t}{T} = 0.8$. The $L2(\Gamma)$ Riesz-representation of the shape derivative is shown as vectors.

$\epsilon = 1$	$\epsilon/2$	$\epsilon/4$	$\epsilon/8$	$\epsilon/16$
$\mathbf{R_2}$	$2.4 \cdot 10^{-2}$	$6.0 \cdot 10^{-3}$	$1.5 \cdot 10^{-3}$	$3.5 \cdot 10^{-4}$
Rate	-	2.00	2.03	2.07

Table: Taylor Test for the shape derivative. The second order residual $R_2 = J(\Omega(\epsilon)[s]) - J(\Omega) - \epsilon \mathrm{d}J(\Omega)[s]$, $s = 0.03\cos(6\pi x)n$, where n is the outward pointing normal vector.

simula