# FEniCS: Hyperelasticity

From documented demonstration number 9 from Dolfin version 1.4.0

Ida Ang (Edited July 16, 2019)

## 1 Potential Energy Minimization

An alternative approach to solving static problems is to consider the minimization of potential energy.

$$\min_{u \in V} \Pi$$

where V is a suitable function space that satisfies the boundary conditions on u.

The total potential energy is given by the sum of the internal and external energy:

$$\begin{aligned} \Pi &= \Pi_{int} + \Pi_{ext} \\ \Pi &= \underbrace{\int_\Omega \psi(u) dx}_{\Pi_{int}} \underbrace{- \int_\Omega B \cdot u dx - \int_{\partial\Omega} T \cdot u ds}_{\Pi_{ext}} \end{aligned} \tag{1}$$

where $\psi$ is the elastic stored energy density, B is body force (per unit reference volume) and T is a traction force (per unit reference area).

Minimization of the potential energy corresponds to the directional derivative of $\Pi$ being zero for all possible variations of u:

$$L(u; v) = D_v \Pi = \frac{d\Pi(u + \epsilon v)}{d\epsilon}\Big|_{\epsilon=0} = 0 \qquad \forall v \in V \tag{2}$$

Note, minimizing $\Pi$ is equivalent to solving the balance of momentum problem. If we use Newton's method, we also want to find the Jacobian of Eq. 2

$$a(u; du, v) = D_{du} L = \frac{dL(u + \epsilon du; v)}{d\epsilon}\Big|_{\epsilon=0} \tag{3}$$

## 2 FEniCS Implementation

### 2.1 Domain and Boundary Conditions

The domain is a unit cube where x is right and y is upwards

$$\Omega = (0, 1) \times (0, 1) \times (0, 1)$$

Create a unit cube with 25 (24 + 1) vertices in one direction and 17 (16 + 1) vertices in the other two directions:

```
mesh = UnitCubeMesh(24, 16, 16)
```

Define a function space with continuous piecewise linear vector polynomials

```
V = VectorFunctionSpace(mesh, "Lagrange", 1)
```

**Boundary Conditions**

Use the following definitions for the boundary conditions, where we have left and right Dirichlet boundary conditions and one Neumann boundary condition:

**Dirichlet**:

$$\Gamma_{D_0} = 0 \times (0,1) \times (0,1)$$

$$\Gamma_{D_1} = 1 \times (0,1) \times (0,1)$$

```
left = CompiledSubDomain("near(x[0], side) && on_boundary", side = 0.0)
right = CompiledSubDomain("near(x[0], side) && on_boundary", side = 1.0)
```

On $\Gamma_{D_0}$, we can define a displacement function to be applied to the right boundary.

$$u = \left[0, \frac{1}{2}\left(\frac{1}{2} + (y - \frac{1}{2})\cos\frac{\pi}{3} - (z - \frac{1}{2})\sin\frac{\pi}{3} - y\right), \frac{1}{2}\left(\frac{1}{2} + (y - \frac{1}{2})\sin\frac{\pi}{3} - (z - \frac{1}{2})\cos\frac{\pi}{3} - x\right)\right]$$

This function gives a twist to the right side of the cube, which is why y and z are specified and x is zero:

```
r = Expression(("scale*0.0",
   "scale*(y0 + (x[1] - y0)*cos(theta) - (x[2] - z0)*sin(theta) - x[1])",
   "scale*(z0 + (x[1] - y0)*sin(theta) + (x[2] - z0)*cos(theta) - x[2])"),
   scale = 0.5, y0 = 0.5, z0 = 0.5, theta = pi/3, degree=2)
```

Note the way we can declare aspects of the expression at the end in the form:

```
Expression("x+y", x = #, y = #, degree = #)
```

On $\Gamma_{D_1}$, we fix the left side with no displacement:

```
c = Constant((0.0, 0.0, 0.0))
```

Combine the left and right boundary conditions with the correct expressions in `bcs`

```
bcl = DirichletBC(V, c, left)
bcr = DirichletBC(V, r, right)
bcs = [bcl, bcr]
```

**Neumann**:

On $\Gamma_N = \frac{\partial\Omega}{\Gamma_D}$, define traction. Define body forces in the y direction

```
T = Constant((0.1, 0.0, 0.0))
```

Define body forces in the y-direction (downwards)

```
B = Constant((0.0, -0.5, 0.0))
```

## 2.2 Kinematics

Define u as the displacement from a previous iteration. Next, find the length of the displacement vector and use that to define the identity tensor.

```
u = Function(V)
d = len(u)
I = Identity(d)
```

Define Eq. 4 and 5, the deformation tensor and the right Cauchy-Green Tensor.

$$F = I + \nabla u \tag{4}$$

Right Cauchy-Green Tensor:

$$C = F^T F \tag{5}$$

Note: `Identity` and `grad` are inbuilt functions.

```
F = I + grad(u)
C = F.T*F
```

## 2.3 Model

An example of a hyperelastic model is the compressible neo-Hookean model:

$$\psi = \frac{\mu}{2}(I_c - 3 - 2\ln J) + \frac{\lambda}{2}\ln\left(J^2\right) \tag{6}$$

where the invariants are described

$$J = \det(F) \quad I_c = \mathrm{tr}(C) \tag{7}$$

and the Lamé coefficients are:

$$\mu = \frac{E}{2(1+\nu)} \quad \lambda = \frac{E\nu}{(1+\nu)(1-2\nu)} \tag{8}$$

and Eq. 6 can be written in terms of the Young's modulus, E, and Poisson's ratio $\nu$ and the Lamé coefficients:

```
E, nu = 10.0, 0.3
```

We can define Eq. 8A, $\mu$, as a constant not an expression because $\nu$ is defined in the code:

```
mu = Constant(E/(2*(1 + nu)))
```

In a similar manner, Eq. 8B, can be defined as a constant because $\nu$ and E are defined in the code. Note that `lambda` is a reserved word so we name the variable`lmbda`

```
lmbda = Constant(E*nu/((1 + nu)*(1 - 2*nu)))
```

Define the invariants Eq. 7 where `tr` and `det` are inbuilt functions:

```
J = det(F)
Ic = tr(C)
```

Write Eq. 6 and 1

```
psi = (mu/2)*(Ic - 3) - mu*ln(J) + (lmbda/2)*(ln(J))**2
Pi = psi*dx - dot(B, u)*dx - dot(T, u)*ds
```

### 2.4 Directional Derivatives

Define the incremental displacement as a trial function and v as a test function:

```
du = TrialFunction(V)
v = TestFunction(V)
```

Directional derivatives are computed of Π (Eq. 2) and L (Eq. 3)
The first directional derivative is analogous to the linear form.

$$L(u;v) = D_v\Pi = \frac{d\Pi(u+\epsilon v)}{d\epsilon}\Big|_{\epsilon=0} = 0$$

```
F = derivative(Pi, u, v)
```

The second directional derivative is analogous to the bilinear form.

$$a(u;du,v) = D_{du}L = \frac{dL(u+\epsilon du;v)}{d\epsilon}\Big|_{\epsilon=0}$$

```
J = derivative(F, u, du)
```

### 2.5 Results

In the first term of the solve function, we want Eq. 2 to equal 0 to minimize the potential energy of the problem. We are solving for `u`, the unknown displacement. `bcs` are the boundary conditions. In the fourth term, we are equating the directional derivative of F to the determinant of F (J = det(F)).

```
solve(F == 0, u, bcs, J=J)
```

Save results

```
file = File("displacement.pvd");
file « u;
```

Don't use the `interactive` command to plot the results. Instead import `matplotlib.pyplot` as `plt`

```
plot(u)
plt.show()
```