FEniCS: Incompressible hyperelasticity with weak form

Ida Ang

1 Problem Definition

This document is based on the original FEniCS hyperelasticity demo, but uses a different formulation. In the original demo, the weak form is not written explicitly and instead the demo uses an energy minimization scheme.

1.1 Energy Density Function

1.1.1 Compressibility

The energy density function for compressible Neo-Hookean materials is:

$$W = \frac{\mu}{2}(I_1 - 3 - 2\ln J) + \frac{\lambda}{2}(\ln J)^2 \tag{1}$$

where μ and λ are Lamé Parameters and can be defined in terms of the Young's modulus, E, and Poisson's ratio ν

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

The first and third invariant of the deformation:

$$I_1 = \mathrm{tr}(\mathbf{C}) \quad I_3 = \mathbf{J} = \det(\mathbf{F}) \tag{3}$$

1.1.2 Incompressibility

For an incompressible material, $\mathbf{J} = \det(\mathbf{F}) = 1$; therefore, Eq. 1 can be modified to:

$$W = \frac{\mu}{2}(I_1 - 3 - 2\ln(1)) + \frac{\lambda}{2}(\ln(1))^2$$
$$W = \frac{\mu}{2}(I_1 - 3)$$

To enforce incompressibility, introduce a lagrange multiplier, p, which acts like a stress term.

$$W(\mathbf{F}) = \frac{\mu}{2}(I_1 - 3) + \mathrm{p}(\mathbf{J} - 1) \tag{4}$$

Now the formulation only uses one Lamé parameter and the first invariant

1.2 Equilibrium equation

P is the nominal stress tensor and b is the body forces:

$$P_{iJ,J} + b_i = 0 \rightarrow \frac{\partial P_{iJ}}{\partial X_J} + b_i = 0 \tag{5}$$

The nominal stress tensor is defined as:

$$P_{iJ} = \frac{\partial W}{\partial F_{iJ}} \tag{6}$$

Start by writing the energy density function Eq. 4 in terms of the invariants

$$W(\mathbf{F}) = \frac{\mu}{2}[\mathrm{tr}(\mathbf{F^T F}) - 3] + p(\det \mathbf{F} - 1) \quad \text{Take derivative with respect to F (Eq. 5)}$$
$$P = \frac{\partial W}{\partial \mathbf{F}} = \frac{\mu}{2}\left(\frac{\partial \,\mathrm{tr}(\mathbf{F^T F})}{\partial \mathbf{F}}\right) + p\frac{\partial \det \mathbf{F}}{\partial \mathbf{F}}$$

Proof of the derivative $\frac{\partial \,\mathrm{tr}(\mathbf{F^T F})}{\partial \mathbf{F}}$ in indicial:

$$\begin{aligned}
\frac{\partial \,\mathrm{tr}(\mathbf{F^T F})}{\partial \mathbf{F}} &= \frac{\partial F_{kI} F_{kI}}{\partial F_{pQ}} \quad \text{Product Rule} \\
&= \frac{\partial F_{kI}}{\partial F_{pQ}} F_{kI} + F_{kI}\frac{\partial F_{kI}}{\partial F_{pQ}} \\
&= \delta_{kp}\delta_{IQ}F_{kI} + F_{kI}\delta_{kp}\delta_{IQ} \\
&= F_{pQ} + F_{pQ} = 2F_{pQ}
\end{aligned}$$

Therefore, we have the following relationship:

$$\frac{\partial \,\mathrm{tr}(\mathbf{F^T F})}{\partial \mathbf{F}} = 2\mathbf{F} \tag{7}$$

Second derivative:

$$\frac{\partial \det \mathbf{F}}{\partial \mathbf{F}} = \det \mathbf{F}\mathbf{F}^{-\mathbf{T}} = \mathbf{J}\mathbf{F}^{-\mathbf{T}}$$

Substituting these relationships in we can obtain the nominal stress tensor, P:

$$\mathbf{P} = \frac{\mu}{2}2\mathbf{F} + \mathrm{p}\mathbf{J}\mathbf{F}^{-\mathbf{T}} = \mu\mathbf{F} + \mathrm{p}\mathbf{J}\mathbf{F}^{-\mathbf{T}} \tag{8}$$

2 Weak Form

Take Eq. 5 and multiply by a test function

$$\begin{aligned}
\frac{\partial P_{iJ}}{\partial X_J}\mathbf{e_i}\cdot v_j\mathbf{e_j} &= -b_i\mathbf{e_i}\cdot v_j\mathbf{e_j} \\
\frac{\partial P_{iJ}}{\partial X_J}v_j\delta_{ij} &= -b_i v_j\delta_{ij} \\
\frac{\partial P_{iJ}}{\partial X_J}v_i &= -b_i v_i \quad \text{Integrate over domain} \\
\int_{\Omega_o}\frac{\partial P_{iJ}}{\partial X_J}v_i dV &= -\int_{\Omega_o} b_i v_i dV
\end{aligned} \tag{9}$$

Use integration by parts:

$$(fg)' = f'g + fg' \rightarrow f'g = (fg)' - fg'$$
$$\frac{\partial P_{iJ}}{\partial X_j} v_i = (P_{iJ} v_i)_{,J} - P_{iJ} v_{i,J}$$

Substitute into Eq. 9:

$$\int_{\Omega_o} (P_{iJ} v_i)_{,J} dV - \int_{\Omega_o} P_{iJ} v_{i,J} dV = -\int_{\Omega_o} b_i v_i dV \quad \text{Use divergence theorem}$$
$$\int_{\partial\Omega_o} P_{iJ} N_J v_i dS - \int_{\Omega_o} P_{iJ} v_{i,J} dV = -\int_{\Omega_o} b_i v_i dV \quad \text{Recognize traction } P_{iJ} N_J = T_i$$
$$\int_{\partial\Omega_o} T_i v_i dS - \int_{\Omega_o} P_{iJ} v_{i,J} dV = -\int_{\Omega_o} b_i v_i dV \quad \text{rearrange}$$
$$\int_{\Omega_o} P_{iJ} v_{i,J} dV = \int_{\Omega_o} b_i v_i dV + \int_{\partial\Omega_o} T_i v_i dS$$

Rewrite in direct notation to obtain weak form:

$$\int_{\Omega_o} P : \mathrm{Grad}(v) dV = \int_{\Omega_o} b v dV + \int_{\partial\Omega_o} T v dS \tag{10}$$

Finally, in order to enforce incompressibility we use Eq. 3 where:

$$\det F = 1 \rightarrow \det F - 1 = 0 \rightarrow J - 1 = 0$$

This can be multiplied by a test function and integrated over the domain Ω_o

$$\int_{\Omega_o} (J-1)\tau dV = 0 \tag{11}$$

3 Bilinear and Linear Form

We have the bilinear form and linear form where we have two test functions

$$a\big((\sigma, u), (\tau, v)\big) = L\big((\tau, v)\big) \qquad \forall (\tau, v) \in \Sigma_0 \times V$$

Therefore, combining Eq. 10 and 11, we have the bilinear and linear forms:

$$a\big((\sigma, u), (\tau, v)\big) = \int_{\Omega_o} \big(P : \mathrm{Grad}(v) + (J-1)\cdot\tau\big) dV \tag{12}$$

$$L\big((\tau, v)\big) = \int_{\Omega_o} b v dV + \int_{\partial\Omega_o} T v dS \tag{13}$$

The bilinear and linear form (Eq. 12 and 13)can also be combined into one statement:

$$a\big((\sigma, u), (\tau, v)\big) - L\big((\tau, v)\big) = \int_{\Omega_o} \big(P : \mathrm{Grad}(v) + (J-1)\cdot\tau\big) dV - \int_{\Omega_o} b v dV - \int_{\partial\Omega_o} T v dS = 0 \tag{14}$$

Solving for two unknowns, displacement (u) and hydrostatic pressure (p), which will allow us to obtain the nominal stress field (P)

4 FEniCS Implementation

First import modules:

```
from dolfin import *
import numpy as np
```

SNES solver parameters are set similarly to the hyperelastic circle contact problem (code not included here)

4.0.1 User Parameters

User parameters allow for control of the number of loop iterations.

```
user_par = Parameters("user")
```

We are ramping the boundary condition displacement from 0 to 0.5 in 5 steps.

```
user_par.add("u_min",0.)
user_par.add("u_max",0.5)
user_par.add("u_nsteps",5)
```

Add user parameters in the global parameter set

```
parameters.add(user_par)
```

Parse parameters from command line `parameters.parse()`

```
info(parameters,True)
user_par = parameters.user
```

4.0.2 Domain

The domain is a unit cube:

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

Create a unit cube with 20 elements with 21 $(20 + 1)$ vertices in one direction and 10 elements with 11 $(10 + 1)$ vertices in the other two directions:

```
mesh = UnitCubeMesh(20, 10, 10)
```

There are two unknowns in our formulation, displacement and hydrostatic pressure (lowercase p) Therefore, define a vector space of degree 2 for displacement (P2) and degree 1 for pressure (P1):

```
P2 = VectorElement("Lagrange", mesh.ufl_cell(), 2)
P1 = FiniteElement("Lagrange", mesh.ufl_cell(), 1)
```

Taylor-Hood elements, `TH`, are elements defined where one vector space is one degree higher than the other. These type of elements are stable for this formulation. In our case, the vector space for displacement is one degree higher than that of pressure:

```
TH = MixedElement([P2,P1])
V = FunctionSpace(mesh, TH)
```

4.0.3 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:

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

On Γ_{D_0}, we can define an initial displacement function to be applied to the right boundary.

$$u = \left[0, s\left(s + (y-s)\cos\frac{\pi}{15} - (z-s)\sin\frac{\pi}{15} - y\right), s\left(s + (y-s)\sin\frac{\pi}{15} - (z-s)\cos\frac{\pi}{15} - x\right)\right]$$

This function gives a twist to the right side of the cube, which is why y and z are specified and x is zero. In this formulation, it is increased to a total theta $= \frac{\pi}{3}$:

```
r = Expression(("scale*0.0",
   "scale*
   (scale + (x[1] - scale)*cos(theta) - (x[2] - scale)*sin(theta) - x[1])",
   "scale*
   (scale + (x[1] - scale)*sin(theta) + (x[2] - scale)*cos(theta) - x[2])"),
   scale = 0, theta = pi/15, degree=2)
```

On Γ_{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))
```

4.0.4 Trial and Test Functions

Define the trial, test functions and unknown functions.

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

Based on our weak forms (Eq. 10 and 11), we have two test functions, v and τ, and two unknowns, displacement and pressure. Therefore, split the test function, v, and unknown function, w. Call v, the test function or displacement v_u and τ, the test function for pressure, v_p. Splitting w will allow us to solve for the unknown displacement and hydrostatic pressure.

```
(v_u, v_p) = split(v)
(u, p) = split(w)
```

4.0.5 Kinematics

First find the spatial dimensions (length) of the displacement tensor
`d = len(u)`

Use dimension (d) to to define the identity tensor. `Identity` is an inbuilt function:
`I = Identity(d)`

Obtain the deformation gradient by first defining the displacement as the difference between the reference and current configuration:

$$\begin{aligned} u &= x - X \quad \text{Take gradient with respect to current configuration} \\ \frac{\partial u}{\partial X} &= \frac{\partial x}{\partial X} - \frac{\partial X}{\partial X} \quad \text{where } F = \frac{\partial x}{\partial X} \\ \nabla u &= F - I \quad \text{Rearrange} \\ I + \nabla u &= F \end{aligned} \tag{15}$$

Right Cauchy-Green Tensor:

$$C = F^T F \tag{16}$$

Define Eq. 15 and 16, the deformation tensor and the right Cauchy-Green Tensor. Note, `grad` is also an inbuilt functions.

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

Define the invariant of the deformation tensor given in Eq. 3, where `det` is a reserved keyword:
`J = det(F)`

4.0.6 Weak Form Definition

Material parameters can be defined in two ways, one more compact, as follows:

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

or

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

We can define Eq. 2, μ, as a constant not an expression because ν is defined in the code:
`mu = Constant(E/(2*(1 + nu)))`

Define Eq. 14 by first defining the nominal stress tensor, P, Eq. 8:

$$P = \mu \mathbf{F} + p\mathbf{J}\mathbf{F}^{-\mathbf{T}}$$

```
def P(u):
   return mu*F + p*J*inv(F.T)
```

Define the weak form (don't worry about the name being the same as the deformation gradient).

$$F = a((\sigma, u), (\tau, v)) + L((\tau, v))$$
$$F = \int_{\Omega_o} \big(P : \mathrm{Grad}(v) + (J-1)\cdot\tau\big) dV - \int_{\Omega_o} bv dV - \int_{\partial\Omega_o} Tv dS = 0$$

Note that body force and traction are constants, the automatic inference algorithm will use 1 Gauss point to represent those integrals. For the first term, the algorithm will estimate the polynomial degree and use many Gauss points to approximate the integral. Using the line `(metadata="quadrature_degree": 4)` allows us to specify the number of Gauss points explicitly:

```
F = (inner(P(u), grad(v_u))
     + inner(J-1., v_p))*dx(metadata="quadrature_degree": 4)
     - dot(B, v_u)*dx - dot(T, v_u)*ds
```

If we don't specify the quadrature degree, a warning will appear about computational time:

Take the directional derivative of F in order to obtain the Jacobian. Recall that w consists of both displacement and hydrostatic pressure, but du refers to the incremental displacement:

$$J = D_{du}F = \frac{dF(u + \epsilon du; v)}{d\epsilon}\Big|_{\epsilon=0} \tag{17}$$

```
J_o = derivative(F, w, du)
```

4.0.7 Solve & Save

Solve the variational problem, where the fourth term specifies the Jacobian of the problem:

```
varproblem = NonlinearVariationalProblem(F, w, bcs, J=J_o)
```

Define the solver and give it both the solver parameters (not the user parameters)

```
solver = NonlinearVariationalSolver(varproblem)
solver.parameters.update(snes_solver_parameters)
```

If info is set to True, solver parameters will be printed out. Solve problem with `solver.solve()`

```
info(solver.parameters, False)
(iter, converged) = solver.solve()
```

Loading parameter (list of values to update displacement)

```
u_list = np.linspace(user_par.u_min,user_par.u_max,user_par.u_nsteps)
```

Save results to an .xdmf file since we have multiple fields

```
file_results = XDMFFile("results.xdmf")
```

Set up the loop to increase displacement. The first statement in the loop updates the expression r with the scaling parameter within u_list

```
for scale in u_list:
     r.scale = scale
     solver.solve()
```

To save the solution, make a deep copy instead of a shallow copy.

```
(u, p) = w.split()
```

Using the rename function, we can specify the name of the variable we are viewing in Paraview:

```
u.rename("Displacement", "u")
p.rename("Pressure", "p")
```

Save in Paraview format:

```
file_results.write(u,scale)
file_results.write(p,scale)
```