Contact Mechanics Problem: Hyperelastic Circle Constrained by Box

Ida Ang (Edited March 20, 2023)

Contents

1 Problem Definition **1**
1.1 Potential Energy Minimization 1
1.2 Kinematics and Specific Strain Energy Density 2

2 FEniCS Implementation **2**
2.1 SNES Solver 2
2.2 TAO solver 3
2.2.1 Class Definition 3

1 Problem Definition

This is a personal document on a contact mechanics problem (currently undocumented demonstration) using the Scalable Nonlinear Equations Solvers (SNES) and Portable, Extensible Toolkit for Scientific Computation (PETSc)'s Toolkit for Advance Optimization (TAO) solvers coded by Corrado Maurini and updated by Tianyi Li (2014). Their version can still be found online, but this version contains my personal modifications and notes. This problem is very similar to documented demonstration #7 hyperelasticity which uses potential energy minimization, an alternative approach to solving static problems.

This example considers a hyperelastic circle under body forces in a box, where $2R = L$ if R is the radius of the circle and L is the length of one dimension of the box. Contact occurs as the circle drops in the box and is in contact with the bottom ($X_1 = -1$) and sides of the cube ($X_2 = -1, 1$). Additionally, changes were made to 1) increment the body force applied, 2) visualize multiple fields on the same mesh, and 3) modify the original box constraint to leave a gap on the bottom. Notation along the lines of Holzapfel is used, (ie bolded vectors and tensors).

1.1 Potential Energy Minimization

Minimization of the energy functional,

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

where V is a suitable function space that satisfies the boundary conditions on the displacement $\mathbf{u}$. The total potential energy, Π, is given by the sum of the internal and external energy:

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

where ψ is the elastic stored energy density, $\mathbf{B}$ is the body force per unit reference volume and $\mathbf{T}$ is a traction force per unit reference area.

Minimization of the potential energy corresponds to the directional derivative of Π being zero for all possible variations of u. (Note, minimizing Π is equivalent to solving the balance of momentum problem.)

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

If we use Newton's method, we also want to find the Jacobian of Eq. 2

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

Note: in the final `solve` function in FEniCS we equate Eq. 3 to $J = \det(\mathbf{F})$

1.2 Kinematics and Specific Strain Energy Density

Note that this is written assuming the reader has a basic understanding of continuum and solid mechanics. The deformation gradient $\mathbf{F}$ and right Cauchy Green (CG) tensor $\mathbf{C}$ are defined as follows

$$\mathbf{F} = \mathbf{I} + \nabla \mathbf{u}$$
$$\mathbf{C} = \mathbf{F}^T \mathbf{F}$$

where the identity tensor is I and the displacement field is $\mathbf{u}$. The first and third invariants of the right CG tensor are defined as follows.

$$I_C = \operatorname{tr} \mathbf{C}$$
$$III_C = \det \mathbf{C}$$

Therefore, we can characterize the volumetric ratio between the current and reference configuration as, where we identify a relationship.

$$J = \det \mathbf{F} \rightarrow J^2 = III_C \tag{4}$$

The stored energy density can be defined with the following equation, which represents

$$\Psi = \frac{\mu}{2}(I_c - 2 - 2\ln J) + \frac{\lambda}{2}(\ln J)^2$$

where the elasticity parameters are defined as the shear modulus, μ (Lame's second parameter) and Lame's first parameter λ. Note that `lambda` is a keyword in FEniCS so we define `lmbda`.

```
E, nu = 10.0, 0.3
mu = Constant(E/(2*(1 + nu)))
lmbda = Constant(E*nu/((1 + nu)*(1 - 2*nu)))
```

where E is the elasticity modulus and ν is Poisson's ratio, a measure of incompressibility.

2 FEniCS Implementation

2.1 SNES Solver

Define SNES solver parameters. `maximum_iterations` sets the maximum newton-raphson iterations the solver will try before exiting. `report` allows for a report of the functional for each iteration as well as a report if the solver fails. `error_on_noncovergence` set to False suppresses the default error message which includes FEniCS contact information.

```
snes_solver_parameters = {"nonlinear_solver":  "snes",
                          "snes_solver":  {"linear_solver":  "lu",
                                           "maximum_iterations":  20,
                                           "report":  True,
                                           "error_on_nonconvergence":  False}}
```

Note that it appears that the PETSc's TAO solver appears to be the best optimization strategy, because higher body forces can be used for larger deformations.

```
problem = NonlinearVariationalProblem(F, u, bc, J=J)
```

Set the boundaries that ensures the circle stays within the box.

```
problem.set_bounds(umin, umax)
```

Call the solver parameters set above in the update statement.

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

If `info` is set to True, information on all solver parameters will be printed

```
info(solver.parameters, True)
```

Stops at this line when the solution diverges
```
(iter, converged) = solver.solve()
```

Warning because modification of the body force results in divergence of the problem. (in the y direction to > -1.0)
```
if not converged:
   warning("This demo is a complex nonlinear problem.  Convergence is not
guaranteed when modifying some parameters or using PETSC 3.2.")
```

Save to an .xdmf file for multiple fields (time step = 0).
```
file = XDMFFile("displacement_snes.xdmf")
file.write(u, 0.)
```

2.2 TAO solver

This file had been edited by Tianyi Li in 2014 and has some edits not only to the solver but to the boundary constraints: Instead of returning an expression for which x[0] must satisfy, the function `near()` returns where x = 0. Functionally, these are equivalent versions of the same definition `symmetry_line`

2.2.1 Class Definition

Define the minimization problem by using the OptimisationProblem class
```
class ContactProblem(OptimisationProblem):
     def __init__(self):
          OptimisationProblem.__init__(self)
```

Objective function
```
     def f(self, x):
          u.vector()[:]  = x
          return assemble(Pi)
```

Define deformation gradient of the objective function
```
     def F(self, b, x):
          u.vector()[:]  = x
          assemble(F, tensor=b)
```

Hessian of the objective function.
```
     def J(self, A, x):
          u.vector()[:]  = x
          assemble(J, tensor=A)
```

The Hessian is related to the Jacobian by the following equation:

$$\mathbf{H}(f(x)) = \mathbf{J}(\nabla f(x))^T$$

Solve the problem by specifying the boundary conditions with u_min and u_max

```
solver.solve(ContactProblem(), u.vector(), u_min.vector(), u_max.vector())
```