FEniCS: Hertzian Contact with a Rigid Indenter Using a Penalty Approach

https://comet-fenics.readthedocs.io/en/latest/demo/contact/penalty.html

Ida Ang (Edited July, 16 2019)

1 Problem Definition

Formulation of frictionless contact between the rigid surface (indenter) and an elastic domain, representing an infinite half-space. Contact will be solved using a penalty formulation allowing a small amount of interpenetration between the solid and the indenter.

Problem uses a linear elastic isotropic constitutive relationship

Indenter Characteristics
The rigid indenter with a spherical surface can be approximated by a parobolic equation instead of explicitly modeled and meshed. Consider the indenter radius, R, to be sufficiently large with respect to the contact region characteristic size (R » a). This relationship, R » a, allows the spherical surface to be approximated by a parabola.

$$h(x,y) = h_o + \frac{1}{2R}(x^2+y^2) \tag{1}$$

where h_o is the initial gap between both surfaces at x = y = 0 (the center of the indenter).

2 Weak Form

Starting from the equilibrium equation:

$$-\operatorname{Div}\sigma = f \rightarrow -\boldsymbol{\nabla}\cdot\sigma = f \tag{2}$$

Write in indicial and convert to weak form

$$\begin{aligned}
-\frac{\partial\sigma_{ij}}{\partial x_j}\mathbf{e_i} &= f_k\mathbf{e_k} \quad \text{Multiply by a test function}\\
-\frac{\partial\sigma_{ij}}{\partial x_j}\mathbf{e_i}\cdot v_p\mathbf{e_p} &= f_k\mathbf{e_k}\cdot v_p\mathbf{e_p}\\
-\frac{\partial\sigma_{ij}}{\partial x_j}v_p\delta_{ip} &= f_k v_p\delta_{kp}\\
-\frac{\partial\sigma_{ij}}{\partial x_j}v_i &= f_k v_k \quad \text{Integrate over the domain}\\
-\int_{\Omega_o}\frac{\partial\sigma_{ij}}{\partial x_j}v_i dV &= \int_{\Omega_o} f_k v_k dV
\end{aligned} \tag{3}$$

Integration by parts on the LHS of Eq. 3

$$(fg)' = f'g + fg' \rightarrow f'g = (fg)' - fg'$$
$$\frac{\partial \sigma_{ij}}{\partial x_j} v_i = (\sigma_{ij} v_i)_{,j} - \sigma_{ij}\frac{\partial v_i}{\partial x_j}$$

Substitute this into Eq. 3

$$\begin{aligned}
-\int_{\Omega_o} (\sigma_{ij}v_i)_{,j} dV + \int_{\Omega_o} \sigma_{ij}\frac{\partial v_i}{\partial x_j} dV &= \int_{\Omega_o} f_k v_k dV \quad \text{Use the divergence theorem}\\
-\int_{\partial\Omega_o} \sigma_{ij} v_i n_j dS + \int_{\Omega_o} \sigma_{ij}\frac{\partial v_i}{\partial x_j} dV &= \int_{\Omega_o} f_k v_k dV \quad \text{Recognize the traction term}\\
-\int_{\partial\Omega_o} t_i v_i dS + \int_{\Omega_o} \sigma_{ij}\frac{\partial v_i}{\partial x_j} dV &= \int_{\Omega_o} f_k v_k dV \quad \text{Rearrange}\\
\int_{\Omega_o} \sigma_{ij}\frac{\partial v_i}{\partial x_j} dV &= \int_{\Omega_o} f_k v_k dV + \int_{\partial\Omega_o} t_i v_i dS
\end{aligned}$$

The principle of virtual work states that:

$$\int_{\Omega} b_i \delta u_i dV + \int_{\partial\Omega} t_i \delta u_i dS = \int_{\Omega} \sigma_{ij}\delta\epsilon_{ij} dV = \int_V \delta W dV \tag{4}$$

Applying this principle to the LHS of our equation where f is equivalent to the body force, b:

$$\begin{aligned}
\int_{\Omega_o} \sigma_{ij}\frac{\partial v_i}{\partial x_j} dV = \int_{\Omega_o} f_k v_k dV + \int_{\partial\Omega_o} t_i v_i dS &= \int_{\Omega} \sigma_{ij}\epsilon_{ij} dV \quad \text{No traction or body forces}\\
0 &= \int_{\Omega} \sigma_{ij}\epsilon_{ij} dV \quad \text{In direct notation}\\
0 &= \int_{\Omega} \sigma(\mathbf{u}) : \epsilon(\mathbf{v}) d\Omega
\end{aligned} \tag{5}$$

2.1 Contact Problem Formulation and Penalty Approach

The unilateral contact condition on the top surface Γ is known as the Hertz-Signorini-Moreau conditions for frictionless contact

$$g \geq 0, \quad p \leq 0, \quad g \cdot p = 0 \quad \text{on } \Gamma \tag{6}$$

Recall that g is the gap between the obstacle surface and the solid surface

$$g = h - u$$

$p = -\sigma_{zz}$ is the pressure.

The simplest way to solve this contact condition is to replace the previous complementary conditions by the following penalized condition:

$$\begin{aligned}
p &= k\langle -g\rangle_+ \quad \text{where } g = h - u\\
&= k\langle u - h\rangle_+
\end{aligned} \tag{7}$$

where k is a large penalizing stiffness coefficient, and we have the definition of the Mackauley bracket

$$\langle x \rangle_+ = \frac{|x| + x}{2} \tag{8}$$

Therefore, we can add the penalty term in Eq. 7 to Eq. 5 after multiplying by the test function

$$\int_\Omega \sigma(\mathbf{u}) : \epsilon(\mathbf{v}) d\Omega + k_{pen} \int_\Gamma \langle u - h \rangle_+ v dS = 0$$

2.2 Bilinear Form and Linear Form

The variational problem can be phrased in the general form.

$$a(u, v) = L(v) \qquad \forall v \in V$$

Rearrange

$$F = a(u, v) - L(v) = 0 \qquad \forall v \in V$$

Therefore

$$F = \int_\Omega \sigma(\mathbf{u}) : \epsilon(\mathbf{v}) d\Omega + k_{pen} \int_\Gamma \langle u - h \rangle_+ v dS = 0 \tag{9}$$

3 FEniCS Implementation

Import modules:

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

Geometry and Domain

The mesh density parameters can be controlled by a number specified in the x and y direction.

```
N = 30
```

Same mesh size in x and y direction, and half in the z (upwards) direction. In Python syntax, N//2, means floor division.

```
mesh = UnitCubeMesh.create(N, N, N//2, CellType.Type_hexahedron)
```

Refine the mesh size to a smaller size closer to the contact area (x = y = z = 0). The indexing [:, :2] refers to the x and y directions where this specifies 0 and 1

```
mesh.coordinates()[:, :2] = mesh.coordinates()[:, :2]**2
```

Refine the mesh where indexing [:,2] refers to the z direction. Use a negative sign to indicate downwards from the point of contact at (x = y = z = 0)

```
mesh.coordinates()[:, 2] = -mesh.coordinates()[:, 2]**2
```

Function Spaces

Define function spaces for displacement and gap respectively using Continuous Galerkin (CG) functions (synonymous with Lagrange specification)

```
V = VectorFunctionSpace(mesh, "CG", 1)
V2 = FunctionSpace(mesh, "CG", 1)
```

Function space for contact pressure defined using Discontinuous Galerkin (DG) for discontinuous basis functions

```
V0 = FunctionSpace(mesh, "DG", 0)
```

Indenter Parameters

Define the indenter or obstacle recalling Eq. 1 which is a parabolic function

$$h(x, y) = h_o + \frac{1}{2R}(x^2 + y^2)$$

This equation requires a definition for h_o, the initial height difference between the indenter and the solid domain, where $h_o = -d$ and R are specified

```
R, d = 0.5, 0.02
```

Define the variables inside the expression class explicitly

```
obstacle = Expression("-d+(pow(x[0],2)+pow(x[1], 2))/(2*R)",
                      d=d, R=R, degree=2)
```

Boundary Condition

Functions for imposition of Dirichlet BC on the boundary. Symmetry conditions are applied to the x = 0 and y = 0 surfaces where for our geometry, these are the back walls

```
def symmetry_x(x, on_boundary):
   return near(x[0], 0) and on_boundary
def symmetry_y(x, on_boundary):
   return near(x[1], 0) and on_boundary
```

The bottom surface is fully fixed (z = -1)

```
def bottom(x, on_boundary):
   return near(x[2], -1) and on_boundary
```

The top surface is defined as a `SubDomain`. This is the part of the domain where the indenter will contact.

```
class Top(SubDomain):
   def inside(self, x, on_boundary):
      return near(x[2], 0.0) and on_boundary
```

Recall that the bottom is fully fixed (z = -1) and therefore has no displacement. Each degree of freedom is constrained making the constant a vector of (0, 0, 0).

```
bc_b = DirichletBC(V, Constant((0.0, 0.0, 0.0)), bottom)
```

Define the symmetry conditions for the boundary (x = y = 0) using the subspaces for function space V for displacement. Since we are utilizing the subspace, we only need to specify a scalar using constant

```
bc_x = DirichletBC(V.sub(0), Constant(0.0), symmetry_x)
bc_y = DirichletBC(V.sub(1), Constant(0.0), symmetry_y)
```

Combine all symmetry conditions and boundary conditions

```
bc = [bc_b, bc_x, bc_y]
```

Isolate facets of the mesh: `"size_t"` gives the topology dimensions of the mesh

```
facets = MeshFunction("size_t", mesh, 2)
```

First initialize the exterior facets of the full surface of the cube to 0, then set the top surface to 1 using the class created

```
facets.set_all(0)
Top().mark(facets, 1)
```

To express integrals over the boundary part using ds, `Measure` will redefine ds in terms of our boundary markers:

```
ds = Measure('ds', subdomain_data=facets)
```

Variational Problem

The trial and test function are defined below. The trial function is the variation of u, du.

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

We are trying to find the displacement, gap function, and contact pressure, which we can name for clearer visualization in Paraview

```
u = Function(V, name="Displacement")
gap = Function(V2, name="Gap")
p = Function(V0, name="Contact pressure")
```

Constitutive Relationship

Define the material parameters: Young's modulus, Poisson's ratio

```
E = Constant(10.)
nu = Constant(0.3)
```

Lamé coefficients

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

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

Define the strain equation

$$\epsilon_{ij} = \frac{1}{2}(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i}) \rightarrow \epsilon = \frac{1}{2}(\nabla u + \nabla u^T) \quad \text{With symmetry}$$

$$\epsilon = \nabla u$$

which can be written as a function definition:

```
def eps(v):
  return sym(grad(v))
```

Define stress using the Neo-Hookean Model in a function definition (Eq. **??**)

$$\sigma_{ij} = \lambda\epsilon_{kk}\delta_{ij} + 2\mu\epsilon_{ij}$$

```
def sigma(v):
    return lmbda*tr(eps(v))*Identity(3) + 2.0*mu*eps(v)
```

Define the Mackauley bracket, Eq. 8 for $\langle x \rangle^+$

$$\langle x \rangle_+ = \frac{|x| + x}{2}$$

```
def ppos(x):
    return (x+abs(x))/2.
```

Define the penalty parameter. A large penalty parameter deteriorates the problem conditioning so that the solving time will drastically increase and the problem can fail:

```
pen = Constant(1e4)
```

Solve

Set up the equation according to Eq. 9 with the penalty parameter

$$F = \int_\Omega \sigma(\mathbf{u}) : \epsilon(\mathbf{v}) d\Omega + k_{pen} \int_\Gamma \langle u - h \rangle_+ v dS = 0$$

h is the obstacle expression defined above. u[2] gives the z direction of displacement. ds(1) denotes integration over subdomain part 1.

```
form = inner(sigma(u), eps(v))*dx
            + pen*dot(v[2], ppos(u[2]-obstacle))*ds(1)
```

Take the directional derivative for the Jacobian

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

Setup the non-linear variational problem

```
problem = NonlinearVariationalProblem(form, u, bc, J=J)
solver = NonlinearVariationalSolver(problem)
```

Setup solver parameters
`cg` stands for the iterative Conjugate-Gradient solver.

```
solver.parameters["newton_solver"]["linear_solver"] = "cg"
```

`ilu` stands for the incomplete lower-upper factorization method.

```
solver.parameters["newton_solver"]["preconditioner"] = "ilu"
solver.solve()
```

Validation & Output

The gap and contact pressure must be projected on the appropriate function spaces in order to evaluate the point-wise (nodal) values.

```
gap.assign(project(obstacle-u[2], V2))
p.assign(-project(sigma(u)[2, 2], V0))
```

The file extensions (.pvd,.vtu) are not suited for multiple fields and parallel writing/reading, but the file extension .xdmf is.

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

The first option saves the output even if the program terminates prematurely. The second output makes sure that multiple outputs within a single time step writes to the same mesh.

```
file_results.parameters["flush_output"] = True
file_results.parameters["functions_share_mesh"] = True
```

Write the results for time step 0

```
file_results.write(u, 0.)
file_results.write(gap, 0.)
file_results.write(p, 0.)
```

3.1 Analytical Solution

The analytical problem gives the following:

- The contact area, a, is of circular shape (d = depth) and radius, R.

$$a = \sqrt{Rd}$$

- The force exerted by the indenter onto the surface is:

$$F = \frac{4}{3}\frac{E}{(1+\nu^2)}ad$$

- The pressure distribution on the contact region is given by (p_o is the maximal pressure):

$$p(r) = p_o\sqrt{1-(\frac{r}{a})^2} \qquad p_o = \frac{3F}{2\pi a^2}$$

These can be solved for in FEniCS

```
a = sqrt(R*d)
F = 4/3.*float(E)/(1-float(nu)**2)*a*d
p0 = 3*F/(2*pi*a**2)
```

Print the maximum pressure and applied force

```
print("Maximum pressure FE: {0:8.5f} Hertz:  {1:8.5f}"
     .format(max(np.abs(p.vector().get_local())), p0))
print("Applied force FE: {0:8.5f} Hertz:  {1:8.5f}"
     .format(4*assemble(p*ds(1)), F))
```

NOTE: The file extension .xdmf will not open if Paraview is outdated (Version 5.5.2 works). An option window should pop up asking to open the XDMF file using three different readers.