Linear Elasticity Problem in 3D

Guhan

Jan. 11, 2018

Objective

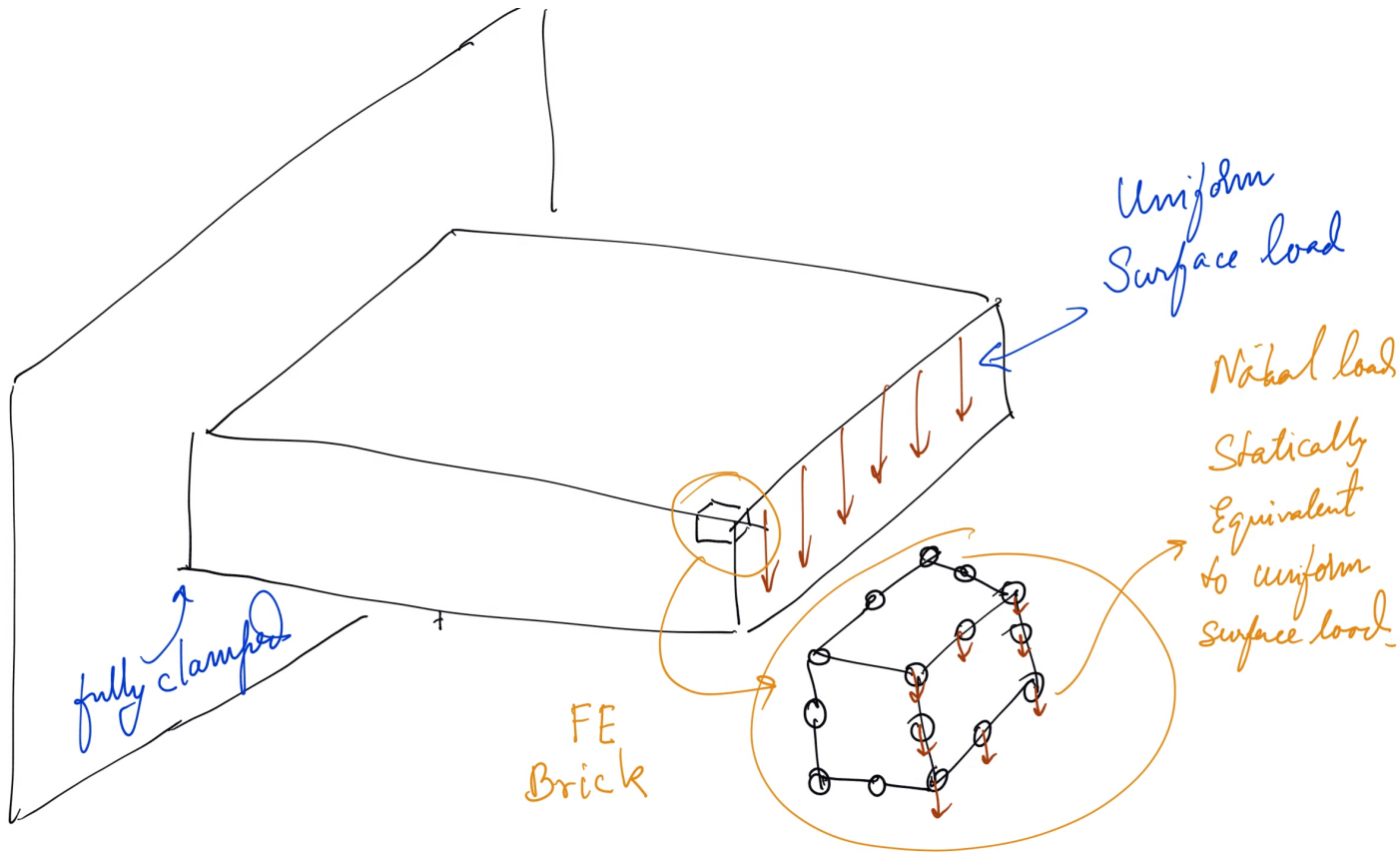
- Solve linear elasticity equation for a test problem using FEniCS
- Compare the result against analytical solution
- Perform convergence study
- Idea is to use this result as a benchmark/validation case for any new codes and methods that we may develop in future

Installing FEniCS

- Installed FEniCS using Docker
- Docker allows for identical deployment across your laptop, the cloud or any high performance computing environments
- Using Macbook for the project
- Visualizing solutions and post processing using Paraview

Problem Description

- Cantilever beam in 3D, fixed at left end with a edge (surface) load at the right end
- Load =1, Stifness=1, Poisson ratio=0.3
- Use 3D Brick elements with midnodes (quadratic interpolation in each direction)



Step 1: Running example problem from FEniCS tutorial

- Example can be found in section 3.3.3 (page 52) of the tutorial
- It is basically modeling a clamped beam deformed under its own weight in 3D
- The code and the results are attached in the next couple of slides

Code from the tutorial - Cantilever beam in 3D under its own weight I

```
1  from __future__ import print_function
2  from fenics import *
3  # Scaled variables
4  L = 1; W = 0.2
5  mu = 1
6  rho = 1
7  delta = W/L
8  gamma = 0.4*delta**2
9  beta = 1.25
10 lambda_ = beta
11 g = gamma
12
13 # Create mesh and define function space
14 mesh = BoxMesh(Point(0, 0, 0), Point(L, W, W), 10, 3, 3)
15 V = VectorFunctionSpace(mesh, 'P', 1)
16
17 # Define boundary condition
18 tol = 1E-14
19
20 def clamped_boundary(x, on_boundary):
21     return on_boundary and x[0] < tol
22
23 bc = DirichletBC(V, Constant((0, 0, 0)), clamped_boundary)
24
25 # Define strain and stress
26
27 def epsilon(u):
28     return 0.5*(nabla_grad(u) + nabla_grad(u).T)
29     #return sym(nabla_grad(u))
```

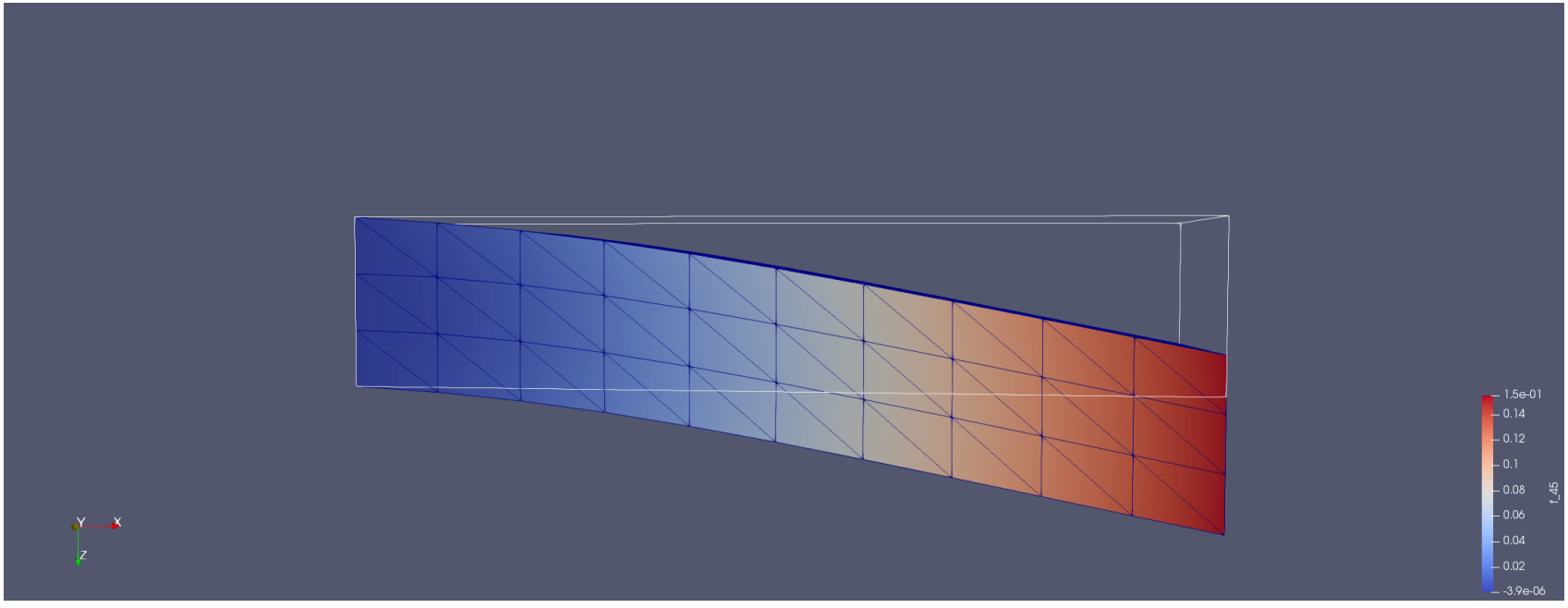
Code from the tutorial - Cantilever beam in 3D under its own weight II

```
30
31  def sigma(u):
32      return lambda_*nabla_div(u)*Identity(d) + 2*mu*epsilon(u)
33
34  # Define variational problem
35  u = TrialFunction(V)
36  d = u.geometric_dimension()  # space dimension
37  v = TestFunction(V)
38  f = Constant((0, 0, -rho*g))
39  T = Constant((0, 0, 0))
40  a = inner(sigma(u), epsilon(v))*dx
41  L = dot(f, v)*dx + dot(T, v)*ds
42
43  # Compute solution
44  u = Function(V)
45  solve(a == L, u, bc)
46
47  # Plot solution
48  plot(u, title='Displacement', mode='displacement')
49
50  # Plot stress
51  s = sigma(u) - (1./3)*tr(sigma(u))*Identity(d)  # deviatoric stress
52  von_Mises = sqrt(3./2*inner(s, s))
53  V = FunctionSpace(mesh, 'P', 1)
54  von_Mises = project(von_Mises, V)
55  plot(von_Mises, title='Stress intensity')
56
57  # Compute magnitude of displacement
58  u_magnitude = sqrt(dot(u, u))
59  u_magnitude = project(u_magnitude, V)
```

Code from the tutorial - Cantilever beam in 3D under its own weight III

```
60  plot(u_magnitude, 'Displacement magnitude')
61  print('min/max u:',
62        u_magnitude.vector().array().min(),
63        u_magnitude.vector().array().max())
64
65  # Save solution to file in VTK format
66  File('elasticity/displacement.pvd') << u
67  File('elasticity/von_mises.pvd') << von_Mises
68  File('elasticity/magnitude.pvd') << u_magnitude
```

Sample result



Step 2: Modifying the example problem - Adding Point/Surface load

- The idea here is to modify the code to suit our problem where a surface load is applied at the end, instead of deforming under its own weight. This would be the first step.

- Once this is successful, then we can go ahead and modify the problem more to include properties like stiffness, poisson ratio and changing the finite element type like using brick element etc.

Step 2.1: Removing body force and adding traction in the code

- Body force is removed

```
f = Constant((0, 0, 0))
```

- Defining traction: arbitrarily choosing -10 as the value here.

```
T = Constant((0, 0, -10))
```

Step 2.2: Defining the boundaries

- A subdomain is defined over which load is applied and integrated later
- Here subdomain is constructed by way of using instance of the subdomain class

```
class Right(SubDomain):
def inside(self, x, on_boundary):
 return near(x[0], L)
```

- If a point is in the region x=L, the it is identified as the right side. Similar instance is used for the left side

```
class Left(SubDomain):
def inside(self, x, on_boundary):
return near(x[0], 0.0)
```

- Initializing sub-domain instances

```
left = Left()
right = Right()
```

Step 2.3: Defining Meshfunction and marking the mesh

- Defining MeshFunction for the boundary domains - not very confident about this part

```
sub_domains = MeshFunction("size_t", mesh)
```

- All the values of MeshFunction is initially set to 0 and then left and right instances are used to mark the regions as 2 and 1 respectively

```
sub_domains.set_all(0)
right.mark(sub_domains, 1)
left.mark(sub_domains, 2)
```

Step 2.4: Applying the load over the right end boundary

- Defining measures associated with the boundaries. Integration over subregions can be specified by measures with different integer labels as arguments

```
ds = Measure("ds")[sub_domains]
```

- Define the variational form using the measures developed. The load or traction is integrated only over the right end subdomain previously marked as 1

```
T = Constant((0, 0, -10))
a = inner(sigma(u), epsilon(v))*dx
L = dot(T, v)*ds(1)
```

Complete code - Cantilever beam in 3D with face load at right end I

```
1
2  from __future__ import print_function
3  from fenics import *
4
5  # Scaled variables
6  L = 1; W = 0.2
7  mu = 1
8  rho = 1
9  delta = W/L
10 gamma = 0.4*delta**2
11 beta = 1.25
12 lambda_ = beta
13 g = gamma
14
15 # Create mesh and define function space
16 mesh = BoxMesh(Point(0, 0, 0), Point(L, W, W), 10, 3, 3)
17 V = VectorFunctionSpace(mesh, 'P', 1)
18
19 # Define boundary condition
20
21 class Left(SubDomain):
22     def inside(self, x, on_boundary):
23         return near(x[0], 0.0)
24
25 class Right(SubDomain):
26     def inside(self, x, on_boundary):
27 #return on_boundary and  abs(x[0]-L) < 0.2 and abs(x[1]-W) < 0.2  and abs(x[1]-W) < 0.2
28         return near(x[0], L)
29
```

Complete code - Cantilever beam in 3D with face load at right end II

```
30  left = Left()
31  right = Right()
32  sub_domains = MeshFunction("size_t", mesh)
33  sub_domains.set_all(0)
34  right.mark(sub_domains, 1)
35  left.mark(sub_domains, 2)
36
37  bc = DirichletBC(V, Constant((0, 0, 0)), left)
38  ds = Measure("ds")[sub_domains]
39  #ds = ds(subdomain_data = MeshFunction)
40
41  # Define strain and stress
42
43  def epsilon(u):
44      return 0.5*(nabla_grad(u) + nabla_grad(u).T)
45      #return sym(nabla_grad(u))
46
47  def sigma(u):
48      return lambda_*nabla_div(u)*Identity(d) + 2*mu*epsilon(u)
49
50  # Define variational problem
51  u = TrialFunction(V)
52  d = u.geometric_dimension()  # space dimension
53  v = TestFunction(V)
54  f = Constant((0, 0, 0))
55
56  #Load at the right end
57  T = Constant((0, 0, -10))
58  a = inner(sigma(u), epsilon(v))*dx
59  L = dot(T, v)*ds(1)
```

Complete code - Cantilever beam in 3D with face load at right end III

```
60
61  # Compute solution
62  u = Function(V)
63  solve(a == L, u, bc)
64
65  # Plot solution
66  plot(u, title='Displacement', mode='displacement')
67
68  # Plot stress
69  s = sigma(u) - (1./3)*tr(sigma(u))*Identity(d)  # deviatoric stress
70  von_Mises = sqrt(3./2*inner(s, s))
71  V = FunctionSpace(mesh, 'P', 1)
72  von_Mises = project(von_Mises, V)
73  plot(von_Mises, title='Stress intensity')
74
75  # Compute magnitude of displacement
76  u_magnitude = sqrt(dot(u, u))
77  u_magnitude = project(u_magnitude, V)
78  plot(u_magnitude, 'Displacement magnitude')
79  print('min/max u:',
80        u_magnitude.vector().array().min(),
81        u_magnitude.vector().array().max())
82
83  # Save solution to file in VTK format
84  # File('elasticity/displacement.pvd') << u
85  # File('elasticity/von_mises.pvd') << von_Mises
86  # File('elasticity/magnitude.pvd') << u_magnitude
```

Possible reasons for code failing

- I am getting segmentation error right now - have to install Valgrind to find any memory leakage
- I am also not confident about the way subdomains and meshfunctions are defined - I am still working on it. I am not sure by setting subdomains at x=0 and x=L, it means the entire face or just the edge. This is where I am digging deep now
- I am not yet clear how the subdomains are integrated into the solver

Other ways to make it work

- One way is to use a third party meshing tool (like Gmsh) and import the meshed geometry into FEniCS

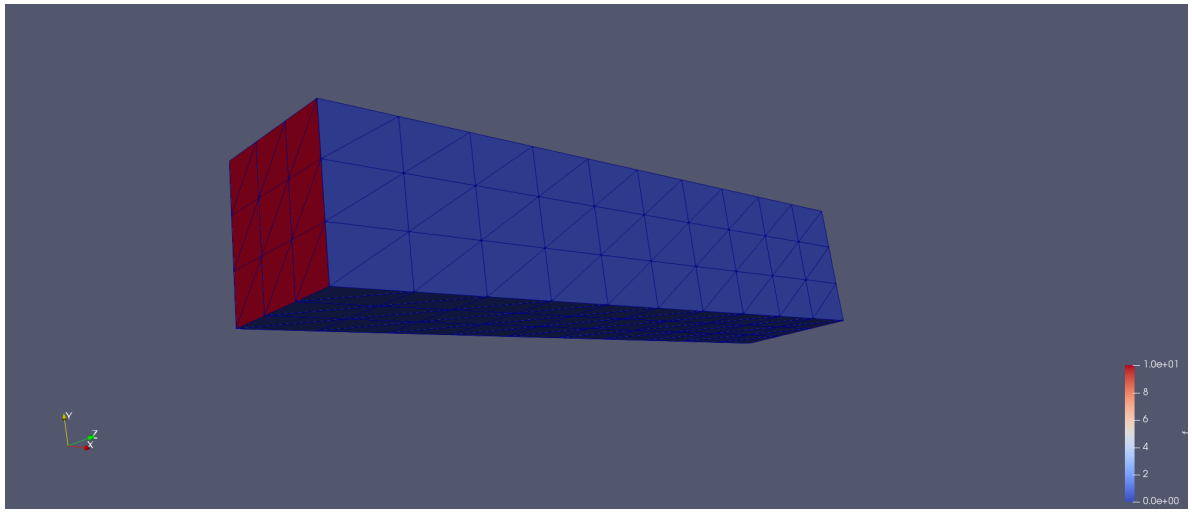
- There are couple of other ways that I found in FEniCS forum - yet to try them all. Will update this space as I learn more

Update - Jan 20: Visualizing Meshes

- I wanted to make sure the boundaries are captured right by visualizing them - made following changes in code

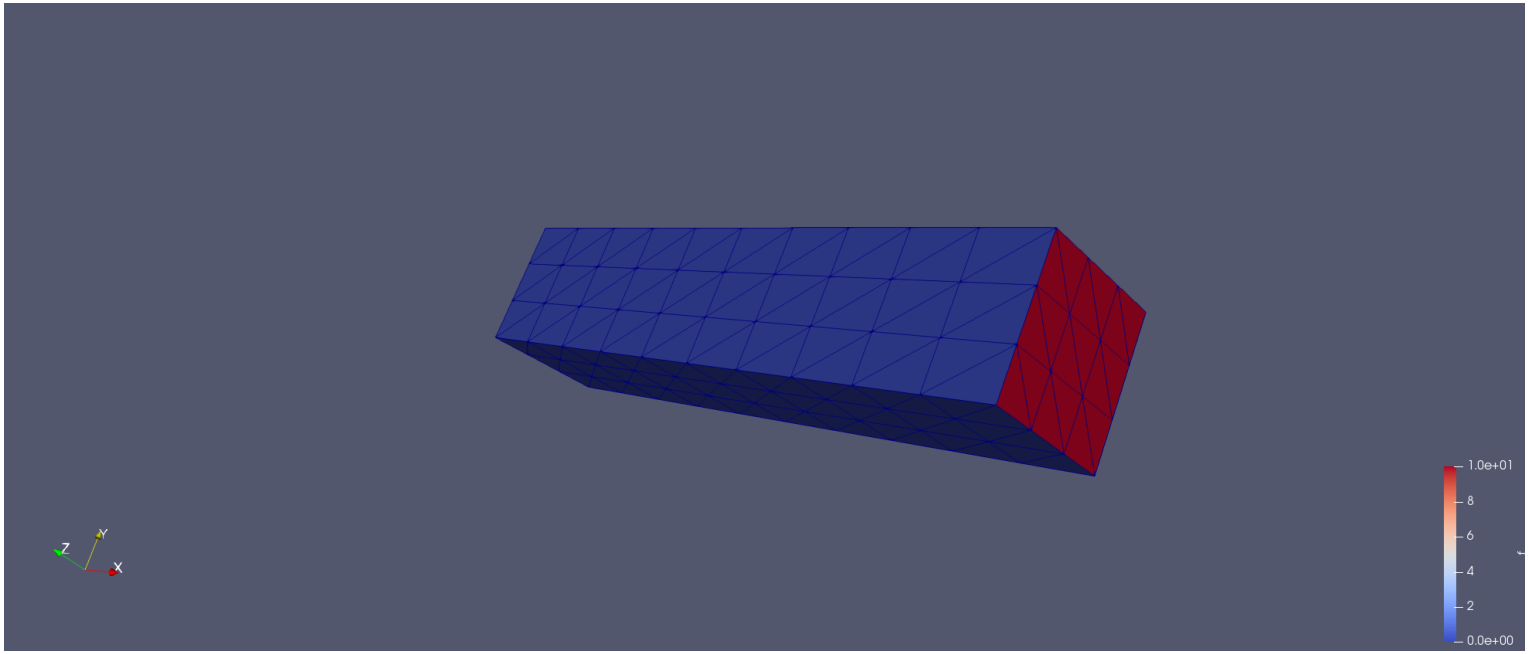
```
sub_domains = FacetFunction("size_t", mesh)
sub_domains.set_all(0)
left.mark(sub_domains, 10)
right.mark(sub_domains,10)
```

- The values are set to 10 just for visualization purpose. Red means 10 and Blue means 0



- Here we see the fixed end at x=0 marked in red. Similarly at x=L, the right end is identified and marked in red.

Update - Jan 20: Visualizing Meshes



- So we can be sure that boundaries are set up right. The main change I made in the code is:

```
sub_domains = FacetFunction("size_t", mesh)
```

- Earlier I was using *MeshFunction* which was wrong

Next step: Visualize the load

- We can now rule out boundaries being the source of error that I get from the code

- The next thing is to check if the load is applied correctly on right end by visualizing them