HOMEWORK 6

CEE 361-513: Introduction to Finite Element Methods

Due: Monday Nov. 26 @ Midnight

NB: Students taking CEE 513 must complete all problems. All other students will not be graded for problems marked with ⋆, but are encourage to attempt them anyhow.

PROBLEM 1:

Read the following sections from the textbook (Hughe's) and summarize the content of each section

1. 1.1-1.9

2. ⋆ 1.10

3. 1.12-1.15

4. 3.6-3.9 (stopping atop of page 148 before the discussion of shape function subroutines)

PROBLEM 2: Learning FEniCS

This problem is to familiarize yourself with FEniCS, a very useful, high-level, and open-source finite element library.

1. Visit the FEniCS webpage (`https://fenicsproject.org`) and go to the Download tab at the top. If you are using Mac or Linux we suggest you use the Anaconda prebuilt distribution. If you are using Windows we suggest you use Docker distribution. If you have any issues along the installation please contact us for help.

2. We are going to solve the simplest possible problem. The problem reads: find $u : [-1, 1] \rightarrow \mathbb{R}$ such that

$$\frac{d^2 u}{dx^2} = f \quad \forall x \in (-1, 1)$$

and

$$u(-1) = g, \quad \frac{du}{dx}(1) = t.$$

We would like to manufacture boundary conditions (t and g) and source terms (f) such that the solution of the above boundary problem is exactly

$$u^e(x) = \frac{1}{2}\cos(2\pi x) + \exp(x) + x^3.$$

To do so we simply set

$$g = u^e(-1), \quad t = \frac{du^e}{dx}(1), \quad f(x) = \frac{d^2 u^e}{dx^2}.$$

With the above we can then derive the weak form of the problem statement to be: find $u \in \mathcal{S}$, with $\mathcal{S}$ being the set of trial functions, such that

$$a(u, v) = F(v) \quad \forall v \in \mathcal{V}$$

where $\mathcal{V}$ is the set of test functions and

$$a(u, v) = \int_{-1}^{1} \frac{du}{dx}\frac{dv}{dx}dx, \quad F(v) = -\int_{-1}^{1} f \, v dx + t \, v(1).$$

All of the above is implemented for you in the attached code named `fenics_truss.py`. Your job for this part of the problem is to go through the code and make sure it runs and it makes sense. Once it does plot the analytical solution u^e and the finite element approximation u^h.

Here all I am going to do is to break down what we are doing.

First and foremost we use `sympy` to create the analytical solution u^e and obtain the expressions for $g, t, f(x)$. We also cast the variable `solution`, which is a `sympy` function, into a `python` lambda function for later plotting.

```
x = sp.symbols('x[0]')                                              1
solution = 1./2*sp.cos( 2.*sp.pi*x ) + sp.exp(x) + x**3             2
ue_code = sp.ccode(solution).replace('M_PI','pi')                   3
t_code = sp.ccode( sp.diff(solution,x,1)).replace('M_PI','pi')      4
f_code = sp.ccode( sp.diff(solution,x,2)).replace('M_PI','pi')      5
solution = sp.lambdify(x,solution)                                  6
```

Next we create the mesh (aka the subdivision) of our domain by first creating a mesh of a domain $[0, 1]$ then scaling the domain by 2 and shifting it to the left by 1.

```
# Start by creating a unit interval mesh              1
# subdivided in ndiv elements                          2
ndiv = 10                                              3
mesh = UnitIntervalMesh(ndiv)                          4
                                                       5
# We are going to shift the above mesh                 6
# such that it dicretizes the interval [-1,1]          7
mesh.coordinates()[:] *= 2                             8
mesh.coordinates()[:] -= 1                             9
```

We now create the subspace of piecewise linear polynomials as we saw in class (the triangle-looking functions also shown in the second plot of the problem on Lagrange polynomials).

```
# Create the function space of                         1
poly_order = 1                                         2
V = FunctionSpace(mesh, 'Lagrange', poly_order)        3
```

Next we identify the parts of the boundary of our domain (in $1 - D$ these are two points, $\Gamma = \{x = -1, x = 1\}$) that are Dirichlet Γ_D and the ones that are Neumann Γ_N.

```
# Define the dirichlet boundary                        1
class dirichlet_boundary(SubDomain):                   2
  def inside(self, x, on_boundary):                    3
    return on_boundary and abs( x + 1. ) < DOLFIN_EPS  4
                                                       5
# Define the neumann boundary                          6
class neumann_boundary(SubDomain):                     7
  def inside(self, x, on_boundary):                    8
    return on_boundary and abs( x - 1. ) < DOLFIN_EPS  9
```

We next mark the boundaries of our domain

```
boundaries = MeshFunction("size_t", mesh, mesh.topology().dim() -1)   1
dirichlet_boundary().mark(boundaries,1)                               2
neumann_boundary().mark(boundaries,2)                                 3
ds = ds(subdomain_data=boundaries)                                    4
```

and we define the Dirichlet boundary conditions

```
bc = DirichletBC(V, ue, dirichlet_boundary())         1
```

Now we are ready to define the trial function and the test function, the expression for the source term, and the Neumann boundary value

```
u = TrialFunction(V)                                   1
v = TestFunction(V)                                    2
```

```
f = Expression(f_code,pi=np.pi, degree=5)              1
t = Expression(t_code,pi=np.pi, degree=5)              2
```

We now define the bilinear form and the forcing functional

```
a = dot(u.dx(0), v.dx(0))*dx                           1
```

```
F  =  -f*v*dx  +  t*v*ds(2)                                                    1
```

and we finally solve the proble

```
uh  =  Function(V)                                                             1
solve(a  ==  F,  uh,  bc)                                                      2
```

3. With the above code construct boundary conditions and source terms such that the analytical solution is of the form

$$u^e(x) = \frac{1}{2}\sin(3\pi x)\,x + \exp(x) + x^2.$$

Then compute the L^2 norm of the error for a mesh with $N = 2^1, 2^2, 2^3, 2^4, \ldots, 2^8$ elements (here you will have to write a `for` loop) and plot the the L^2 norm of the error against the average element size $h = 2/2^1, 2/2^2, \ldots, 2/2^8$ on a log-log plot (use `matplotlib loglog` plot).

4. What is the rate of convergence of the solution? Namely, we see that for a mesh size h the L^2 norm of the error is

$$\|e^h\|_{L^2} = \|u^h - u^e\| \approx \alpha h^k$$

where α is a constant independent of h. Our goal is to determine k.

To do so choose a fixed h (for example $h = 2/2^7$) then look at

$$\frac{\|e^h\|_{L^2}}{\|e^{h/2}\|_{L^2}} \approx \frac{\alpha\, h^k}{\alpha\,(h/2)^k} = 2^k \Rightarrow k = \log_2\frac{\|e^h\|_{L^2}}{\|e^{h/2}\|_{L^2}}.$$

5. What is the rate of convergence if instead of linear polynomials we use quadratic or cubic polynomials? Namely, repeat the above steps by changing the `poly_order` variable in

```
poly_order  =  1                                                               1
V  =  FunctionSpace(mesh,  'Lagrange',  poly_order)                            2
```

Note, that when we change the polynomial order we are simply replacing the "triangle" functions with the higher order Lagrange polynomials of the previous problem constructed over the interior of each element.