PRECEPT 6

CEE 361-513: Introduction to Finite Element Methods

Monday Nov. 6

SECTION 1: Introduction to FEniCS

Here we would provide a brief introduction to FEniCS starting with installation and then going over the code provided as part of homeowrk 5.
FEniCS could be installed by following the instructions at : `https://fenicsproject.org/download/`

- Windows users are encouraged to install FEniCS on Docker.
- Linux and MAC users are encouraged to install FEniCS on Anaconda.

If your installation was correct you should be able to perform the following:

1. Download the Intro to Fenics code from course website.
2. Run the code and obatin the following plot:

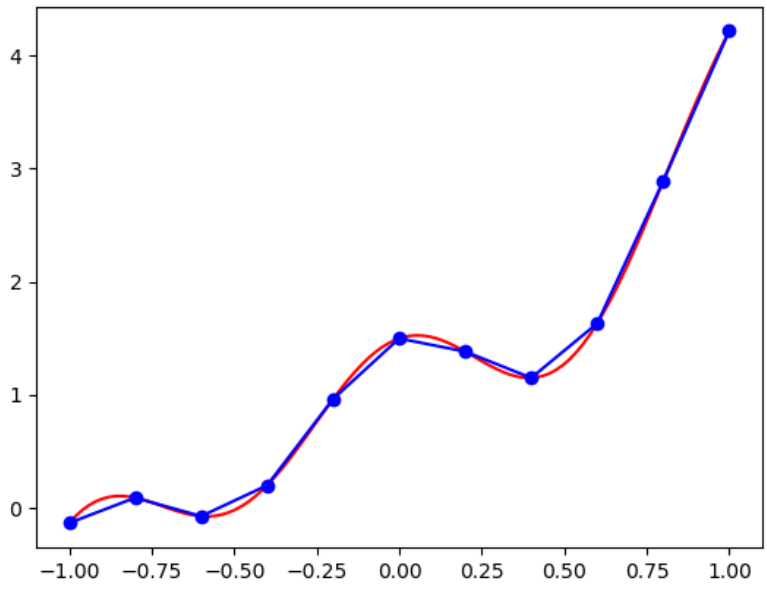

Figure 1: Plot of exact and FEM solution

- FEniCS is an open-source platform for solving partial differential equation.
- FEniCS converts the scientific model into efficient finite element code. The FFC just-in-time (JIT) compiler message you saw while running the code does the same. During this process, C++ code is generated by FFC (FEniCS from compiler) and compiled by C++ compiler.

1.1 Code Overview

We first create a manufactured solution to our problem as mentioned in the homework. Here we have used sympy to create the solution symbolically and converted it to C++ code to be later read.

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

Next part we create a domain and subdivide the domain. This process is called creating the mesh.

```
ndiv = 10
mesh = UnitIntervalMesh(ndiv)
```

The mesh we have created is a Unit length line whose end points are 0 and 1. The problem has the domain between $[-1, 1]$. Hence, we elongate and shift the mesh next.

```
mesh.coordinates()[:] *= 2
mesh.coordinates()[:] -= 1
```

After defining our domain we need to define the function space. The function space we are using has Lagrange polynomials as the interpolating functions and the order of the polynomials is 1. You can play around with this number.

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

Next part of the code defines classes for the Dirichlet and Neumann boundaries. The boundaries for a 1-D problem are essentially points.

```
# Define the dirichlet boundary
class dirichlet_boundary(SubDomain):
  def inside(self, x, on_boundary):
    return on_boundary and abs( x + 1. ) < DOLFIN_EPS

# Define the neumann boundary
class neumann_boundary(SubDomain):
  def inside(self, x, on_boundary):
    return on_boundary and abs( x - 1. ) < DOLFIN_EPS
```

Next we would like to mark the different boundaries so that the code can differentiate between the two. 'ds' is the measure which stores this information.

```
boundaries = FacetFunction("size_t", mesh)
dirichlet_boundary().mark(boundaries,1)
neumann_boundary().mark(boundaries,2)
ds = ds(subdomain_data=boundaries)
```

Next part involves using the analytical solution to define the boundary condition. In the boundary condition we are essentially applying the value of the analytical solution (**ue**) on the boundary as our boundary condition. The step involved creating a FEniCS function of our analytical solution. The analytical solution has the symbol "pi" which we need to explain what it represents to FEniCS. And this expression is interpolated on the domain we need to tell what is the degree of the interpolation we need. We have passed a high value because the analytical solution is highly non-liner.

```
# Define the analaytical solution
ue = Expression(ue_code, pi=np.pi, degree=5)

```

```
# Define the dirichlet boundary conditions
bc = DirichletBC(V, ue, dirichlet_boundary())
```

Then we create the test and trial function on the function space.

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

Now, as was done with the analytical solution we create the expressions for the forcing terms.

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

The next part is creating the bilinear form which you obtain from the weak form of the PDE you derived. Following that we create the right hand side

```
# Define the bilinear form
a = dot(u.dx(0), v.dx(0))*dx

# Define the forcing
F = -f*v*dx + t*v*ds(2)
```

Next we solve the problem and compute errors and plot the solution.

SECTION 2: Intro to GMSH

In the above examples of we had used simple meshes. A 1-D element and a circle. For real life problems you may need to use complicated meshes. One of the mesh generating softwares available for use is GMSH.

- Freely available 3-D finite element mesh generator.
- Simple to use GUI.
- Input can be done through the GUI or ASCII text files.
- Can be downloaded from `http://www.gmsh.info/#Download`
- Highly useful if you want to use complicated geometry which are beyond the inbuilt capabilities of FEniCS. Further, FEniCS currently only supports triangular/tetrahedral meshes. If you need to obtain a quadrilateral/hexahedral element you can use GMSH

We would show how to create a simple square mesh with a hole and import the same in FEniCS for use. Follow the following steps:

1. Open GMSH
2. From the right panel, select Geometry → Elementary entities → Add → Point
3. Add the following corner points $[(0.0, 0.0), (1.0, 0.0), (1.0, 1.0), (0.0, 1.0)]$ of the square. Add the center and four points on the perimeter for the hole $[(0.5, 0.5), (0.4, 0.5), (0.6, 0.4), (0.5, 0.6), (0.5, 0.4)]$

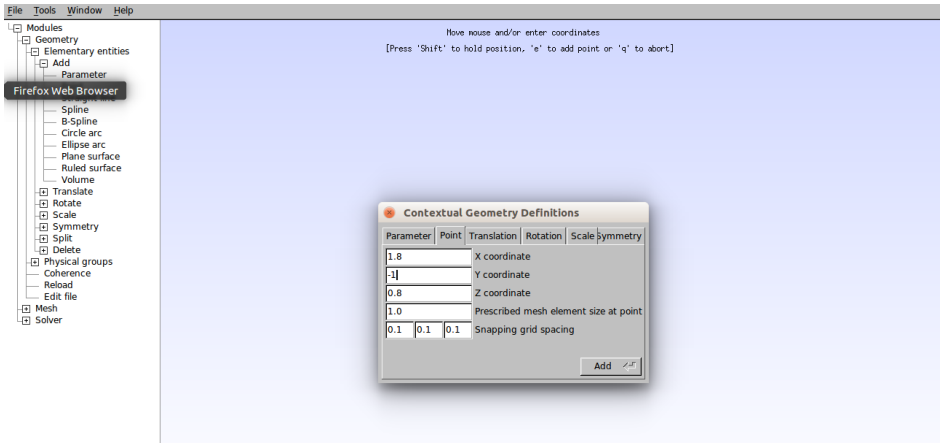

Figure 2: Adding the points

4. Click on Add → Straight Line. Then connect the points as lines

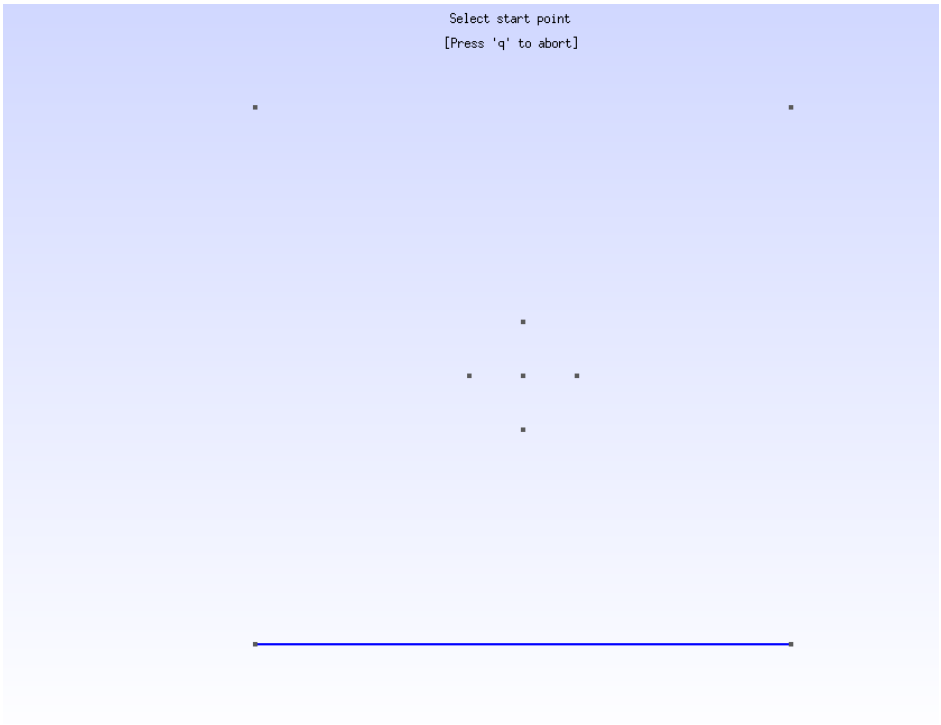

Figure 3: Adding the lines

5. Next click on Circle arc, and connect the points in the center to create circular arcs.
6. The initial geometry looks like the following.

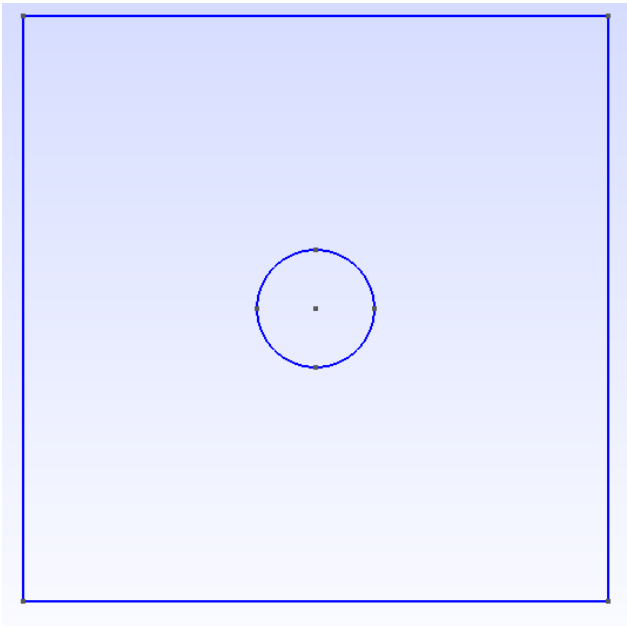

Figure 4: Initial geometry

7. Next click on Plane Surface from the same panel and select the outer boundaries and the holes when directed. This should result in the following:

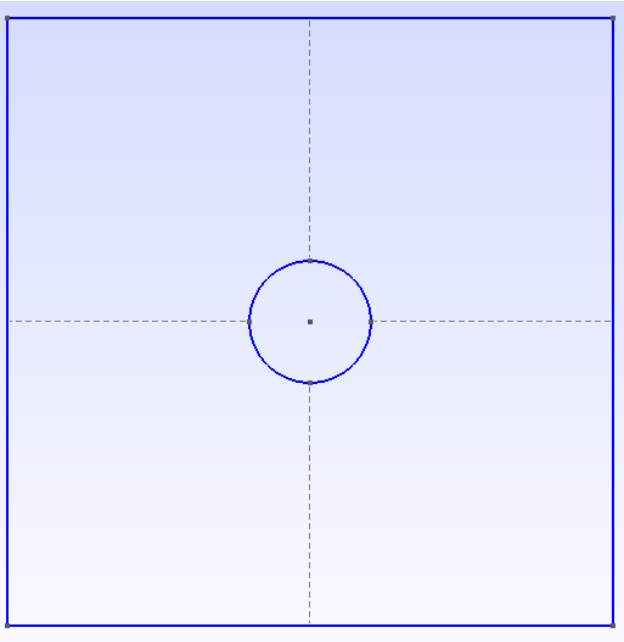

Figure 5: Initial geometry

8. Click on Physical groups under Elementary entities and select Line. Click on all the lines in the geometry
9. Next select Surface under Physical groups and click on the surface.
10. Finally click Mesh $\rightarrow$ 2D and you should have obtained a coarse mesh as below:

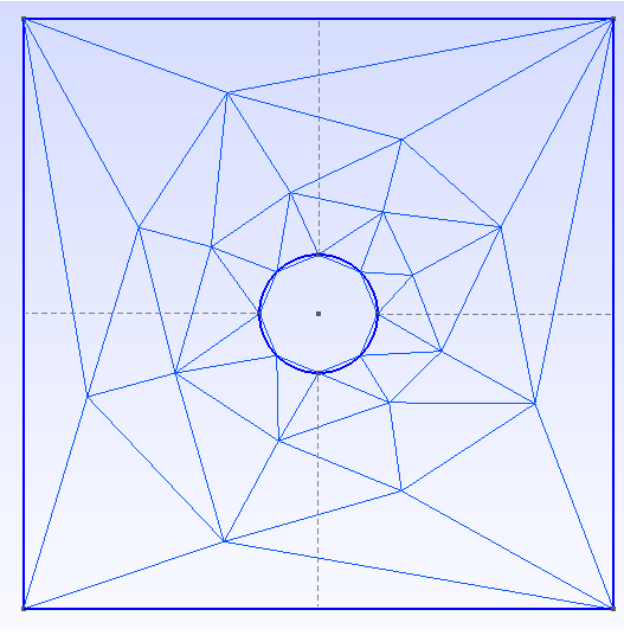

Figure 6: Coarse Triangular Mesh

11. We can refine the mesh by selecting Refine by splitting to obtain a refined mesh.

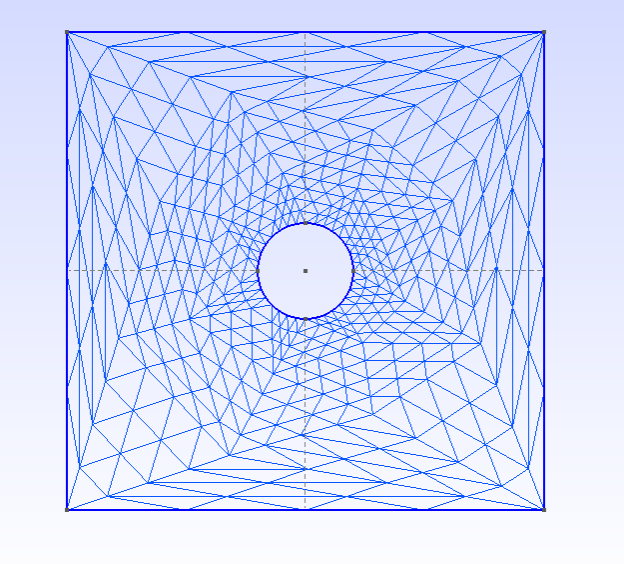

Figure 7: Refined Triangular Mesh

12. You can change the meshing to quadrilateral elements by clicking on the mesh and selecting All mesh options. Then select General and check the box which says "Recombine all triangular meshes". You would then obtain a quad mesh as below:

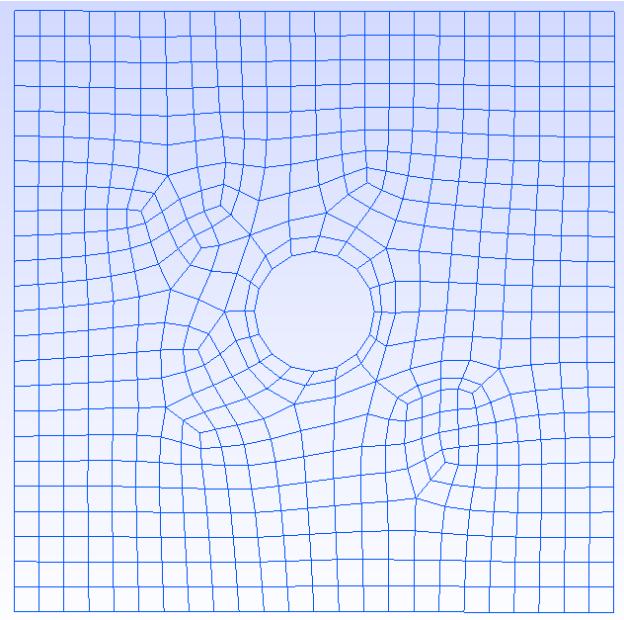

Figure 8: Refined Quad Mesh

13. Lastly you can save the mesh

Once the mesh is saved we need to import it in FEniCS. Reminder that quad elements are not supported in FEniCS yet, so you cannot import that mesh.

1. Let us say the mesh was saved as square_hole.msh

2. In terminal, perform:

    - `python`
    - `from dolfin import *`
    - `import os`
    - `os.system('dolfin-convert square_hole.msh square_hole.xml')`
      This should create a mesh which we can import to dolfin as:
    - mesh = Mesh("square_hole.xml")

SECTION 3: Intro to ParaView

Here we would briefly go over a Visualization software Paraview, save the mesh obtained before as XDMF format and view it in ParaView. A few pointers:

- Freely available, open-source data visualization software.
- Supports formats such as VTK(.vtu), XDMF(.xdmf), ParaView Files (.pvd)
- Can be downloaded from `https://www.paraview.org/download/`

Next we save the mesh obtained before as xdmf format.

1. Navigate in terminal to the folder where you saved the mesh 'square_hole.xml'.
2. Perform:
   (a) `python`
   (b) from dolfin import *
   (c) mesh = Mesh("square_hole.xml")
3. Write the mesh as XDMF using `XDMFFile("square_hole.xdmf").write(mesh)`
4. You must have created two files .xdmf and .h5. The HDF(.h5) file stores and organizes the data. You can view the mesh by opening the xdmf file in Paraview. The first output looks like:

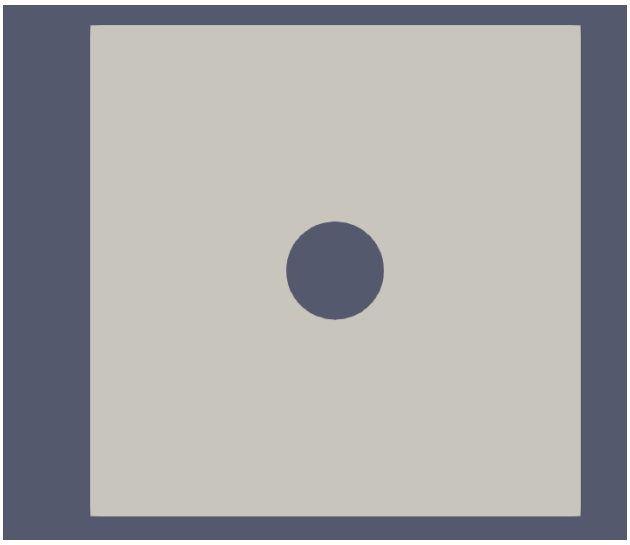

Figure 9: Surface view in ParaView

5. Change the view to Surface With Edges and you can view the mesh.

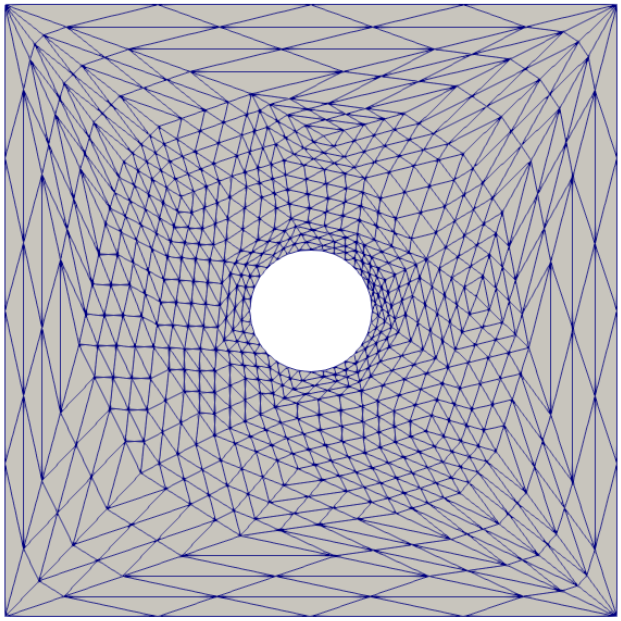

Figure 10: Surface with edges view in ParaView