PRECEPT 8

CEE 361-513: Introduction to Finite Element Methods

Monday Nov. 20

PROBLEM 1: Solving 2-D Problem in FEniCS

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

1. We are going to solve the simplest 2-D problem, the Poisson Equation. The problem reads: find $u : [0,1] \times [0,1] \to \mathbb{R}$ such that

$$\begin{aligned} -\Delta u &= f \quad \text{in } \Omega \\ u &= 0 \quad \text{on } \Gamma_D \end{aligned}$$

We solve the problem on the domain $\Omega = [0,1] \times [0,1]$. Γ_D represents the Dirichlet boundary of the domain. For this problem we are given that:

$$f(\boldsymbol{x}) = 10\sin(10x_1) + 5\cos(5x_2)$$
$$\Gamma_D : \text{Boundary}$$

The weak form for the above problem is simply:

$$\int_\Omega \nabla u \cdot \nabla v \; dx = \int_\Omega f v \; dx$$

The first step is creating the mesh:

```
# Start by creating a unit square mesh
# subdivided in ndiv elements
ndiv = 10
mesh = UnitSquareMesh(ndiv, ndiv)
```

The mesh looks like:

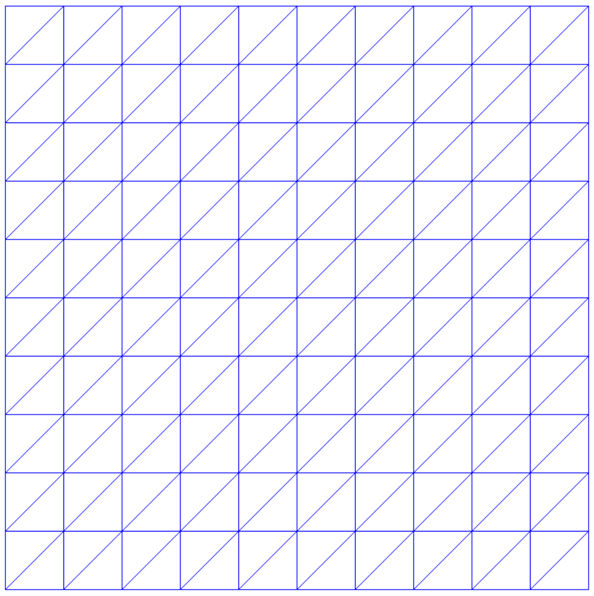


Figure 1: The Mesh for the poisson problem

As you can see the mesh has triangular elements. FEniCS currently does not support quadrilateral elements. If you need quad elements, you can use other mesh generating softwares such as GMSH. We now create the subspace of piecewise linear polynomials as we saw in class.

```
# Create the function space of polynomial order of the
# lagrange interpolating function
poly_order = 1
# Added space dim, here we solve the problem in 2-D
space_dim = 2

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

Next we identify the parts of the boundary of our domain (in $2-D$ these are two lines, $\Gamma = \{x = 0, x = 1\}$) that are Dirichlet Γ_D. We do not have a Neumann Boundary in this problem

```
# The domain is clamped at x=0 and x=1
# Define the dirichlet boundary
class dirichlet_boundary(SubDomain):
  def inside(self, x, on_boundary):
    return on_boundary
```

and we define the Dirichlet boundary conditions

```
# Define the dirichlet boundary conditions
# Since it is clamped we assign the value of zero
bc = DirichletBC(V, Constant(0.0), dirichlet_boundary())
```

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

```
# Define the trial and test function
u = TrialFunction(V)
v = TestFunction(V)
```

```
# Define the forcing function
f = Expression('(10.0*sin(10.0*x[0])+5.0*cos(5.0*x[1]))', degree=5)
```

We now define the bilinear form and the forcing functional

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

```
# Define the forcing
F = f*v*dx
```

and we finally solve the proble

```
uh = Function(V, name='displacement')
solve(a == F, uh, bc)
```

Now we can plot the solution:

```
plot(uh, interactive=True)
```

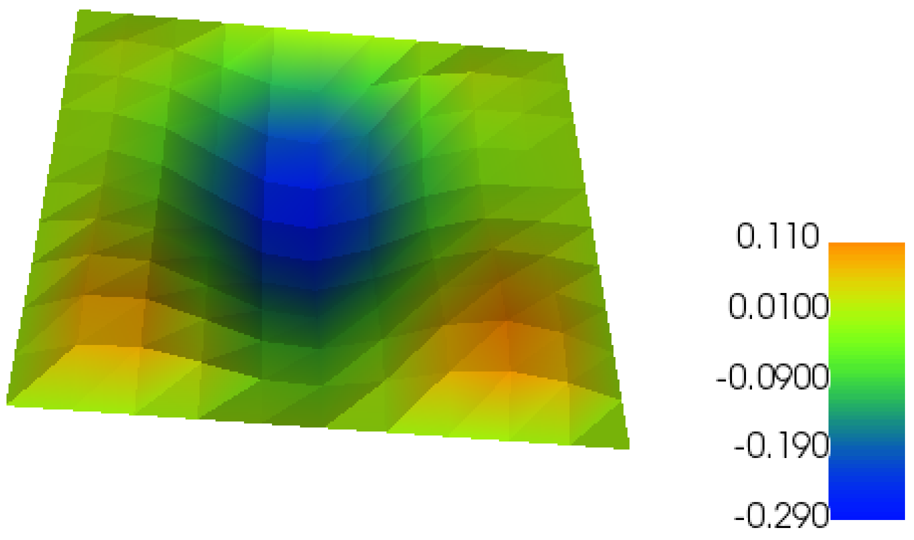


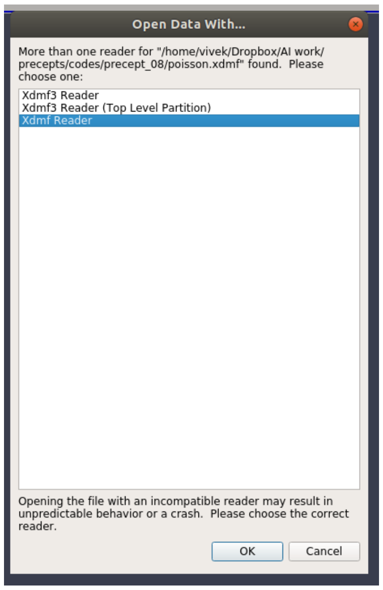
Figure 2: Solution plotted in the inbuilt viewer

2. Saving the solution in VTK and XDMF formats Most of the times it is not convenient to plot the solution in the inbuilt viewer. The viewer has limited capabilities. We would use the Visualization tool Paraview to view our solution.
   The first step is saving the file in required formats. You can do this in two ways:

   ```
   # Save as VTK format
   file_vtk = File('poisson.pvd')
   file_vtk << uh

   # Save as xdmf format
   file_xdmf = XDMFFile('poisson.xdmf')
   file_xdmf.write(uh,0)
   ```

   Next you can open the files in Paraview. Let us open the xdmf file. When prompted to select the viewer select XDMF Reader.


Open Data With...

More than one reader for "/home/vivek/Dropbox/AI work/precepts/codes/precept_08/poisson.xdmf" found. Please choose one:

Xdmf3 Reader
Xdmf3 Reader (Top Level Partition)
Xdmf Reader

Opening the file with an incompatible reader may result in unpredictable behavior or a crash. Please choose the correct reader.

Figure 3: XDMF options for viewing

You would have obtained a plot like this:

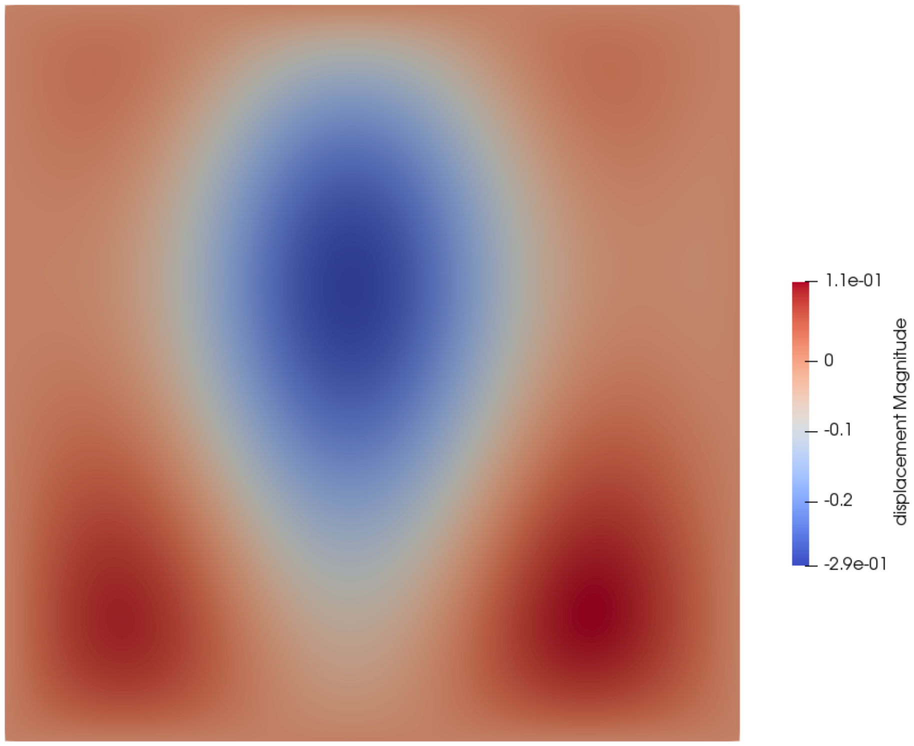


Figure 4: Solution plotted in Paraview

Even though this provides all the information required we can do better in terms of visualization. Perform the following steps:

(a) Click on 'Calculator' button.

(b) In the calculator, multiply the solution ('displacement') with kHat.
Now, the solution can be visualized as a deformed shape.

(c) Click on 'Warp By Vector' button. This would have deformed the solution in z-direction.

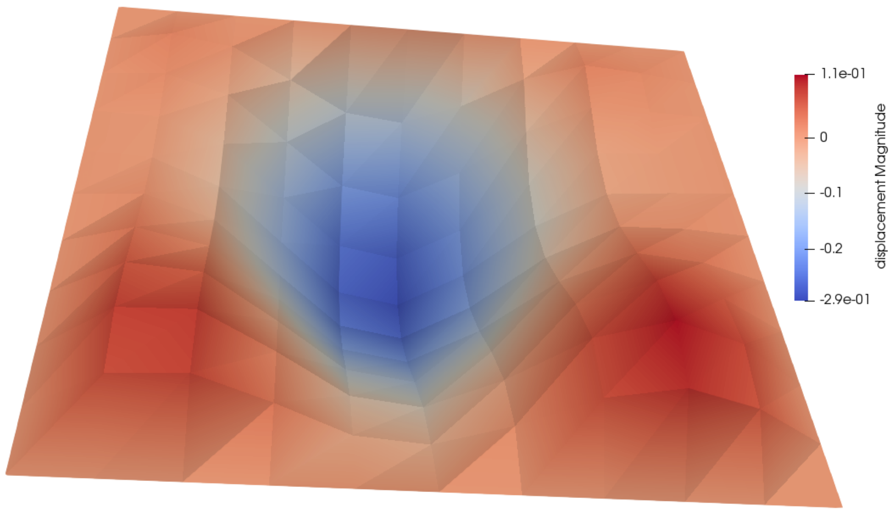


Figure 5: Deformed Shape using 'Warp By Vector' function in Paraview

PROBLEM 2: Python Modules

We briefly introduce the use of python modules and packages. We have already used modules in homework 5 and 6. We had used modules such as `lagrange_polynomials.py`. As the code gets longer it becomes unmanageable if everything is defined in a single file. Hence, by using modules we break the codes into many files which we can later call as required. This also helps to re-use the code we have we already written efficiently. We can improve on this by collecting relevant module files in separate directories and import them from those directories. This extension of module mechanism to directories is called "Package". The idea is to have all related module files together in a package.
To tell python that a certain directory is a package, you need a file names `__init__.py` in the directory. We can then import all or some modules from the package as required.
Consider the following directory structure:

```
FEM_project/
    main.py
    setup_system/
        __init__.py
        geometry.py
        mesh.py
        ..
    solve_system/
        __init__.py
        assemble_system.py
        apply_bc.py
        ..
```

The file `main.py` could be importing all or some of the modules. It could have commands such as:
`from setup_system import *`
When the python interpreter encounters the above line it would execute the following:

1. Looks for the `__init__.py` file in the directory.
2. Performs all the top-level statements in the file `__init__.py`
3. Imports everything from the directory

We could also have a command such as: `from setup_system import geometry`

1. Looks for the `__init__.py` file in the directory.
2. Performs all the top-level statements in the file `__init__.py`
3. Imports the file geometry