FEniCS: Poisson Problem

From documented demonstration number 17 from Dolfin version 1.4.0

Ida Ang (Last edit: July 1, 2019)

1 Problem Definition

Poisson's equation with Neumann boundary conditions for domain Ω and boundary $\partial\Omega = \Gamma_D \cup \Gamma_N$

$$\begin{aligned} -\nabla^2 u &= f(x,y) && \text{in } \Omega \\ u &= 0 && \text{on } \Gamma_D \\ \nabla u \cdot n = \frac{\partial u}{\partial n} &= g && \text{on } \Gamma_N \end{aligned} \tag{1}$$

We can rewrite the Neumann boundary condition in indicial:

$$\begin{aligned} \nabla u \cdot n &= \frac{\partial u_j}{\partial x_i}\big(\mathbf{e_j} \otimes \mathbf{e_i}\big) \cdot n_k \mathbf{e_k} \\ &= \frac{\partial u_j}{\partial x_i} n_k \mathbf{e_j}(\mathbf{e_i} \times \mathbf{e_k}) \\ &= \frac{\partial u_j}{\partial x_i} n_k \mathbf{e_j} \delta_{ik} \\ \nabla u \cdot n &= \frac{\partial u_j}{\partial x_i} n_i \mathbf{e_j} = g_j \mathbf{e_j} \end{aligned} \tag{2}$$

Therefore the Neumann boundary Condition can be written as

$$\frac{\partial u_j}{\partial x_i} n_i = g_j \tag{3}$$

2 Variational Weak Form

Start by writing the laplacian in indicial notation:

$$\begin{aligned} \nabla u &= \frac{\partial u_j}{\partial x_i} \mathbf{e_j} \otimes \mathbf{e_i} \\ \nabla \cdot \nabla u &= \frac{\partial^2 u_j}{\partial x_i \partial x_k}\big(\mathbf{e_j} \otimes \mathbf{e_i}\big) \cdot \mathbf{e_k} = \frac{\partial^2 u_j}{\partial x_i \partial x_k} \mathbf{e_j} \delta_{ik} \\ \nabla \cdot \nabla u &= \frac{\partial^2 u_j}{\partial x_i^2} \mathbf{e_j} \end{aligned}$$

Write Eq. 1 in indicial notation:

$$-\frac{\partial^2 u_j}{\partial x_i^2} \mathbf{e_j} = f_k \mathbf{e_k} \tag{4}$$

Multiply Eq. 4 by test function, v:

$$
\begin{aligned}
-\frac{\partial^2 u_j}{\partial x_i^2}\mathbf{e_j} \cdot v_p \mathbf{e_p} &= f_k \mathbf{e_k} \cdot v_p \mathbf{e_p} \\
-\frac{\partial^2 u_j}{\partial x_i^2} v_j &= f_k v_k \quad \text{Integrate over domain } \Omega \\
-\int_\Omega \frac{\partial^2 u_j}{\partial x_i^2} v_j dx &= \int_\Omega f_k v_k dx
\end{aligned} \tag{5}
$$

Use Integration by Parts:

$$
(fg)' = f'g + fg' \rightarrow f'g = (fg)' - fg' \rightarrow f''g = (f'g)' - f'g' \tag{6}
$$

Where we can substitute $f'' = \frac{\partial^2 u_j}{\partial x_i^2}$ and $g = v_j$ into Eq. 6

$$
\frac{\partial^2 u_j}{\partial x_i^2} v_j = \left(\frac{\partial u_j}{\partial x_i} v_j\right)_{,i} - \frac{\partial u_j}{\partial x_i}\frac{\partial v_j}{\partial x_i}
$$

Change signs and substitute into Eq. 5

$$
\begin{aligned}
-\int_\Omega \left(\frac{\partial u_j}{\partial x_i} v_j\right)_{,i} dx + \int_\Omega \frac{\partial u_j}{\partial x_i}\frac{\partial v_j}{\partial x_i} dx &= \int_\Omega f_k v_k dx \quad \text{Use divergence theorem on first term} \\
-\int_{\partial\Omega} \frac{\partial u_j}{\partial x_i} v_j n_i ds + \int_\Omega \frac{\partial u_j}{\partial x_i}\frac{\partial v_j}{\partial x_i} dx &= \int_\Omega f_k v_k dx \quad \text{recognize Eq. 3 on first term} \\
-\int_{\Gamma_N} g_j v_j ds + \int_\Omega \frac{\partial u_j}{\partial x_i}\frac{\partial v_j}{\partial x_i} dx &= \int_\Omega f_k v_k dx \quad u = 0 \text{ for Dirichlet boundary} \\
\int_\Omega \frac{\partial u_j}{\partial x_i}\frac{\partial v_j}{\partial x_i} dx &= \int_\Omega f_k v_k dx + \int_{\Gamma_N} g_j v_j ds
\end{aligned} \tag{7}
$$

Weak form in direct notation:

$$
\int_\Omega \nabla u \cdot \nabla v dx = \int_\Omega f v dx + \int_{\Gamma_N} g v dx \tag{8}
$$

3 Bilinear Form and Linear Form

We can rewrite eq. 8 in terms of the bilinear form, a(u,v), and linear form, L(v).

$$
a(u, v) = L(v) \qquad \forall v \in \hat{V} \tag{9}
$$

Therefore:

$$
a(u, v) = \int_\Omega \nabla u \cdot \nabla v dx \tag{10}
$$

$$
L(v) = \int_\Omega f v dx + \int_{\Gamma_N} g v dx \tag{11}
$$

4 FEniCS Implementation

Import Dolfin module and module for plotting

```
from dolfin import *
import matplotlib.pyplot as plt
```

4.1 Domain and Boundaries

Domain is a unit square:

$$\Omega = [0, 1] \times [0, 1]$$

We can use the inbuilt function `UnitSquareMesh` subdivided into 32 elements

```
mesh = UnitSquareMesh(32, 32)
```

The space `V` consists of first-order, continuous Lagrange finite element functions

```
V = FunctionSpace(mesh, "Lagrange", 1)
```

Complete boundary

$$\partial\Omega = \Gamma_D \cup \Gamma_N$$

Neumann boundary - On the bottom and top of the square

$$\Gamma_N = (x, 0) \cup (x, 1) \subset \partial\Omega$$

Dirichlet boundary - On the sides of the square

$$\Gamma_D = (0, y) \cup (1, y) \subset \partial\Omega$$

Since the Dirichlet BC is applied to the right and left sides of the square we can define a boundary using a machine precision value, `DOLFIN_EPS`. This sets x = 0 and x = 1

```
def boundary(x):
     return x[0] < DOLFIN_EPS or x[0] > 1.0 - DOLFIN_EPS
```

On the Dirichlet boundary, u = 0, which can be applied to the boundary defined above using the class `DirichletBC`:

```
u0 = Constant(0.0)
bc = DirichletBC(V, u0, boundary)
```

Note

- `u0` is a scalar not a vector because the function space is first-order
- The Neumann boundary condition is expressed in the weak form of the Poisson problem. Dirichlet boundary conditions have to be defined explicitly outside of the weak form.

4.2 Definitions of input functions

The fluid source term is defined below:

$$f = 10 \exp\big(-((x - 0.5)^2 + (y - 0.5)^2)/0.02\big) \tag{12}$$

Normal derivative:

$$g = \sin(5x) \tag{13}$$

Write the defined input functions: Eq. 12 and 13 in C++ syntax (more efficient). The degree is specified for interpolation on the discretized mesh.

```
f = Expression("10*exp(-(pow(x[0] - 0.5, 2) + pow(x[1] - 0.5, 2)) / 0.02)",
degree=2)
g = Expression("sin(5*x[0])", degree=2)
```

4.3 Solve

Define the trial and test (unknown we are solving for) function:
```
u = TrialFunction(V)
v = TestFunction(V)
```

Define the bilinear, Eq. 10, and linear equation, Eq. 11:
```
a = inner(grad(u), grad(v))*dx
L = f*v*dx + g*v*ds
```

Note: FEniCS knows that `ds` specifies the whole boundary. The reason that we can specify the full boundary is because the test function v equals zero on the Dirichlet boundaries on the sides of the unit square.

In this statement, u stores the solution as a function of the space V (originally u is defined as the trial function)
```
u = Function(V)
```

The default solver for the function solve is usually lower-upper (LU) decomposition. Different solvers can be specified depending on the type of problem.
```
solve(a == L, u, bc)
```

The file can be saved and renamed for visualization in Paraview
```
file = File("poisson.pvd")
u.rename("Displacement", "u")
file « u
```

The file can also be immediately plotted using the imported module matplotlib
```
plot(u)
plt.show()
```