FEniCS: Mixed formulation for Poisson Equation

From documented demonstration number 12 from Dolfin version 1.4.0

Ida Ang (Edited July 1, 2019)

1 Problem Definition

Consider a model for the temperature u in a body occupying a domain Ω subject to a heat source, f. Let $\sigma = \sigma(x)$ denote heat flux. It follows by conservation of energy that the outflow of energy over the boundary $\delta\Omega$ must be balanced by the energy emitted by the heat source f:

$$\begin{aligned} \int_{\delta\Omega} \sigma \cdot n ds &= -\int_{\Omega} f dx \quad \text{Use Guassian divergence theorem} \\ \int_{\Omega} \nabla \cdot \sigma dx &= -\int_{\Omega} f dx \\ \nabla \cdot \sigma &= -f \quad \text{in } \Omega \end{aligned} \tag{1}$$

Assume that the heat flux, σ, is proportional to the gradient of the negative temperature u (Fourier's law)

$$\begin{aligned} \sigma &= -\kappa \nabla u \quad \text{Heat conductivity } \kappa = -1 \\ \sigma &= \nabla u \\ \sigma - \nabla u &= 0 \quad \text{in } \Omega \end{aligned} \tag{2}$$

The same equations arise in connection with flow in porous media, and are referred to as Darcy flow.

Boundary conditions:
The Dirichlet boundary condition is the natural boundary condition within the variational form. The Neumann boundary condition for flux is now the essential boundary condition which must be enforced in the function space.

$$\text{Dirichlet Boundary: } u = u_o \quad \text{on } \Gamma_D \tag{3}$$

$$\text{Neumann Boundary: } \sigma \cdot n = g \quad \text{on } \Gamma_N \tag{4}$$

2 Weak Form

We have two partial differential equations, Eq. 1 and 2. This leads to the introduction of two different test functions, τ and v.

First

Convert Eq. 2 to indicial notation:

$$\begin{aligned}
\sigma = \nabla u \rightarrow \sigma_{ij}(\mathbf{e_i} \otimes \mathbf{e_j}) &= \frac{\partial u_i}{\partial x_j}\big(\mathbf{e_i} \otimes \mathbf{e_j}\big) \quad \text{Multiply with test function } \tau \\
\sigma_{ij}(\mathbf{e_i} \otimes \mathbf{e_j}) \cdot \tau_k \mathbf{e_k} &= \frac{\partial u_i}{\partial x_j}\big(\mathbf{e_i} \otimes \mathbf{e_j}\big) \cdot \tau_k \mathbf{e_k} \\
\sigma_{ij}\tau_k \mathbf{e_i}\delta_{jk} &= \frac{\partial u_i}{\partial x_j}\tau_k \mathbf{e_i}\delta_{jk} \qquad (5) \\
\sigma_{ij}\tau_j \mathbf{e_i} &= \frac{\partial u_i}{\partial x_j}\tau_j \mathbf{e_i} \quad \text{Integrate over domain} \\
\int_\Omega \sigma_{ij}\tau_j \mathbf{e_i} dx &= \int_\Omega \frac{\partial u_i}{\partial x_j}\tau_j \mathbf{e_i} dx
\end{aligned}$$

Use integration by parts on the right hand side

$$(fg)' = f'g + fg' \rightarrow f'g = (fg)' - fg' \qquad (6)$$

Where we can substitute $f' = \frac{\partial u_i}{\partial x_j}\mathbf{e_i}$ and $g = \tau_j$ into Eq. 6

$$\frac{\partial u_i}{\partial x_j}\mathbf{e_i}\tau_j = (u_i \mathbf{e_i}\tau_j)_{,j} - u_i \mathbf{e_i}\frac{\partial \tau_j}{\partial x_j}$$

Substitute into RHS of Eq. 5

$$\begin{aligned}
\int_\Omega \sigma_{ij}\tau_j \mathbf{e_i} dx &= \int_\Omega (u_i \mathbf{e_i}\tau_j)_{,j} dx - \int_\Omega u_i \mathbf{e_i}\frac{\partial \tau_j}{\partial x_j} dx \quad \text{Use divergence theorem} \\
\int_\Omega \sigma_{ij}\tau_j \mathbf{e_i} dx &= \int_{\partial\Omega} u_i \mathbf{e_i}\tau_j n_j ds - \int_\Omega u_i \mathbf{e_i}\frac{\partial \tau_j}{\partial x_j} dx \quad \text{Rearrange} \\
\int_\Omega \sigma_{ij}\tau_j \mathbf{e_i} dx + \int_\Omega \frac{\partial \tau_j}{\partial x_j} u_i \mathbf{e_i} dx &= \int_{\partial\Omega} \tau_j n_j u_i \mathbf{e_i} ds
\end{aligned}$$

Convert this into direct notation:

Term 1

$$\sigma \cdot \tau = \sigma_{ij}(\mathbf{e_i} \otimes \mathbf{e_j}) \cdot \tau_k \mathbf{e_k} = \sigma_{ij}\tau_k \mathbf{e_i}(\mathbf{e_j} \cdot \mathbf{e_k}) = \sigma_{ij}\tau_k \mathbf{e_i}\delta_{jk} = \sigma_{ij}\tau_j \mathbf{e_i}$$

Term 2

$$(\nabla \cdot \tau)u = \Big(\frac{\partial \tau_j}{\partial x_i}\mathbf{e_j} \cdot \mathbf{e_i}\Big)u_k \mathbf{e_k} = \frac{\partial \tau_j}{\partial x_i}\delta_{ji}u_k \mathbf{e_k} = \frac{\partial \tau_j}{\partial x_j}u_k \mathbf{e_k}$$

Term 3

$$(\tau \cdot n)u = (\tau_j \mathbf{e_j} \cdot n_k \mathbf{e_k})u_i \mathbf{e_i} = \tau_j n_k \delta_{jk} u_i \mathbf{e_i} = \tau_j n_j u_i \mathbf{e_i}$$

Finally, we have the weak form:

$$\int_\Omega \sigma \cdot \tau dx + \int_\Omega (\nabla \cdot \tau)u dx = \int_\Gamma (\tau \cdot n)u ds \qquad \forall \tau \in \Sigma \qquad (7)$$

Second
Write Eq. 1 in indicial

$$\nabla \cdot \sigma = -f \rightarrow \frac{\partial \sigma_{ij}}{\partial x_j}\mathbf{e_i} = -f_i\mathbf{e_i} \quad \text{Multiply by test function } v$$
$$\frac{\partial \sigma_{ij}}{\partial x_j}\mathbf{e_i} \cdot v_p\mathbf{e_p} = -f_i\mathbf{e_i} \cdot v_p\mathbf{e_p}$$
$$\frac{\partial \sigma_{ij}}{\partial x_j}v_p\delta_{ip} = -f_iv_p\delta_{ip}$$
$$\frac{\partial \sigma_{ij}}{\partial x_j}v_i = -f_iv_i \quad \text{Integrate over domain}$$
$$\int_\Omega \frac{\partial \sigma_{ij}}{\partial x_j}v_i dx = -\int_\Omega f_iv_i dx$$

Change to direct notation, leaving the following variational (weak) formulations. Looking at Eq. 7 we can see that the Dirichlet boundary condition (Eq. 3) is a natural boundary condition, which should be applied to the variational form.

$$\int_\Omega \sigma \cdot \tau dx + \int_\Omega (\nabla \cdot \tau)u dx = \int_\Gamma (\tau \cdot n)u_o ds \qquad \forall \tau \in \Sigma \tag{8}$$

$$\int_\Omega (\nabla \cdot \sigma)v dx = -\int_\Omega fv dx \qquad \forall v \in V \tag{9}$$

3 Bilinear Form and Linear Form

Inserting the boundary conditions, this variational problem can be phrased in the general form. Recall our general form for the demonstration of the simple Poisson problem was written as follows:

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

For this problem, we can modify this general form because we have two test functions

$$a\big((\sigma, u), (\tau, v)\big) = L\big((\tau, v)\big) \qquad \forall (\tau, v) \in \Sigma_0 \times V$$

We can rewrite Eq. 8 and 9 in the bilinear form and linear form:

$$a\big((\sigma, u), (\tau, v)\big) = \int_\Omega \Big[\sigma \cdot \tau + (\nabla \cdot \tau)u + (\nabla \cdot \sigma)v\Big] dx \tag{10}$$

$$L\big((\tau, v)\big) = -\int_\Omega fv dx + \int_{\Gamma_D} (\tau \cdot n)u_o ds \tag{11}$$

Note: The Neumann boundary condition for the flux (Eq. 4) is now an essential boundary condition, which should be enforced in the function space. The Dirichlet boundary condition is the natural boundary condition within the weak form.

4 FEniCS Implementation

4.1 Domain

The domain is a unit square:

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

Create a unit square

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

To discretize the formulation, two discrete function spaces $\Sigma_h \subset \Sigma$ and $V_h \subset V$ are needed to form a mixed function space $\Sigma_h \times V_h$.

$$W = \{(\tau, v) \text{ such that } \tau \in BDM, v \in DG\}$$

Define a mixed function space:
Brezzi-Douglas-Marini (BDM) is a vector space for stress and Discontinuous Galerkin (DG) is a scalar space for displacement. A stable choice of finite element spaces is to let Σ_h be the BDM elements of polynomial order k, 1, and let V_h be DG elements of polynomial order k-1, 0.

```
BDM = FiniteElement("BDM", mesh.ufl_cell(), 1)
DG = FiniteElement("DG", mesh.ufl_cell(), 0)
```

Note: Depreciated function where `W = BDM * DG` can't be used

```
W = FunctionSpace(mesh, BDM * DG)
```

Next we need to specify the trial functions (unknowns) and the test functions. Note `TrialFunctions` for more than one `TrialFunction`. We are finding the stress, σ, and displacement, u corresponding to the two `TestFunctions`, τ and v.

```
(sigma, u) = TrialFunctions(W)
(tau, v) = TestFunctions(W)
```

Define input functions

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

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

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 = 1)
g = Expression("sin(5.0*x[0])", degree = 1)
```

4.2 Boundary

Dirichlet boundary (Natural in this formulation): right and left sides of the unit square

$$\Gamma_D = \{(0, y) \cup (1, y) \in \partial\Omega\}$$

Neumann boundary (Essential in this formulation): bottom and top sides of the unit square

$$\Gamma_N = \{(x, 0) \cup (x, 1) \in \partial\Omega\}$$

The displacement on the Dirichlet BC is specified as zero ($u_o = 0$), changing the linear form, Eq. 11. The bilinear form remains the same.

$$a\big((\sigma, u), (\tau, v)\big) = \int_\Omega \Big[\sigma \cdot \tau + (\nabla \cdot \tau)u + (\nabla \cdot \sigma)v\Big] dx$$
$$L\big((\tau, v)\big) = -\int_\Omega f v dx$$

Define variational form

```
a = (dot(sigma, tau) + div(tau)*u + div(sigma)*v)*dx
L = - f*v*dx
```

Prescribe the boundary condition for the flux. Specifying the relevant part of the boundary can be done similarly to the Poisson demo, except we are defining the Neumann bc not the Dirichlet bc:

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

Subclassing Expression Class

An essential bc is handled by replacing degrees of freedom (DOF) by the DOF evaluated at the given data. The BDM finite element spaces are vector-valued spaces and hence the degrees of freedom act on vector-valued objects. Construct a G:

$$G \cdot n = g \rightarrow G = gn$$

```
class BoundarySource(UserExpression):
   def __init__(self, mesh, **kwargs):
      self.mesh = mesh
```

Overloading the `eval_cell` method, instead of the usual `eval`, allows us to extract more geometry information such as the facet normals (cell.normal()).

```
   def eval_cell(self, values, x, ufc_cell):
      cell = Cell(self.mesh, ufc_cell.index)
      n = cell.normal(ufc_cell.local_facet)
```

Note, how g is defined again within this class as well as within the expression

```
      g = sin(5*x[0])
```

Construct g (G = gn). according to Eq. 13 and index values to return.

```
      values[0] = g*n[0]
      values[1] = g*n[1]
```

Since this is a vector-valued expression, we also need to overload the `value_shape` method.

```
   def value_shape(self):
      return (2,)
```

Next, call the constructed class `BoundarySource` within the main code:

```
G = BoundarySource(mesh, degree=2)
```

We want to apply the boundary condition to the first subspace of the mixed space. Subspaces

of a mixed FunctionSpace can be accessed by the method sub. In our case, this reads `W.sub(0)`

```
bc = DirichletBC(W.sub(0), G, boundary)
```

4.3 Solution and Output

We need to create a `Function` to store the solutions. The (full) solution will be stored in "w", which we initialise using FunctionSpace "W"

```
w = Function(W)
```

We equate the bilinear and linear forms and call the boundary condition. The computation is performed by calling `solve`.

```
solve(a == L, w, bc)
```

The separate components `sigma` and `u` of the solution can be extracted by the `split` function.

```
(sigma, u) = w.split()
```

Save each component by using .xdmf file format which can save multiple fields. Use `rename` function so the output won't be saved under a random name.

```
file = XDMFFile("mixed_poisson.xdmf")
sigma.rename("Stress", "sigma")
u.rename("Temperature", "u")
```

Parameters that will allow the file to be saved even if the script exits prematurely and will allow the functions to share the same mesh.

```
file.parameters["flush_output"] = True
file.parameters["functions_share_mesh"] = True
```

Save parameters with time point 0.0

```
file.write(sigma, 0.0)
file.write(u, 0.0)
```