lm2d_3d1d

February 26, 2019

1 Coupling PDEs on 3d-1d domains with Lagrange multipliers

We consider two Poisson problems posed on domains Ω_3 and Ω_1 such that $\Omega_1 \subset \Omega_3$ and $|\Omega_1| \ll |\Omega_3|$ and which are coupled by Dirichlet and Neumann constraints over the common manifold Γ

$$\begin{aligned} -\Delta u_3 &= f_3 \quad \text{in } \Omega_3, \\ -\Delta u_1 &= f_1 \quad \text{in } \Omega_1, \\ \nabla u_3 \cdot n - \nabla u_1 \cdot n &= 0 \qquad \text{on } \Gamma, \\ u_3 - u_1 &= g \qquad \text{on } \Gamma. \end{aligned}$$

Posing certain assumptions on the solutions and the geometry it is possible to formulate the problem as a $3d$-$1d$ coupled problem where newly u_1 is defined on a curve γ which is the centerline of Γ.

1.1 Coupling via a 1d Lagrange multiplier

The coupling can be accomplished through a Lagrange multiplier p which is defined on γ. In this case the Dirichlet constraint uses an averaging operator Π such that

$$(\Pi u_3)(x) = |C(x)|^{-1} \int_{C(x)} u_3 dl,$$

where $C(x)$ is the curve obtained by intersecting Γ with a plane passing though x and having normal collinear with tangent to γ at x. The Lagrange multiplier formulation of the coupled problem then leads to the operators

$$\mathcal{A} = \begin{pmatrix} -\Delta_3 & & \Pi' \\ & -\Delta_1 & -I \\ \Pi & -I & \end{pmatrix}$$

Support for this type of coupling has existed in FEniCS_ii for almost a year now.

1.2 Coupling via a 2d Lagrange multiplier

Alternatively, the coupling can be accomplised via a Lagrange multiplier defined on a manifold. Note that the manifold may be different than Γ but here we shall for simplicity assume that they are identical. In this case the Dirichlet constraint on Γ requires two "reduction" operators T and E. By trace T the $3d$ solution can be evaluated at the manifold, while E extends u_1 to Γ. The operator form of the coupled problem then reads

$$\mathcal{A} = \begin{pmatrix} -\Delta_3 & & T' \\ & -\Delta_1 & -E' \\ T & -E & \end{pmatrix}$$

Support for this type of coupling is a new (experimental) feature of FEniCS_ii.

1.3 Seting up geometry and mesh

In [1]:
```
from xii import EmbeddedMesh, Trace, Extension, ii_assemble, ii_convert, ii_Function
from weak_bcs.la_solve import direct_solve_superlu
from weak_bcs.bc_apply import apply_bc
from weak_bcs.utils import block_form
from dolfin import *


ncells = 16
mesh_3d = UnitCubeMesh(ncells, ncells, 2*ncells)  # 3d mesh

f = MeshFunction('size_t', mesh_3d, 1, 0)
CompiledSubDomain('near(x[0], 0.5) && near(x[1], 0.5)').mark(f, 1)
# Mesh to extend from (1d mesh)
mesh_1d = EmbeddedMesh(f, 1)

n = ncells + 1
A, B = 1./ncells*(n/2-1), 1./ncells*(n/2+1)

f = MeshFunction('size_t', mesh_3d, 2, 0)
f_str = ['((near(x[0], A) || near(x[0], B)) && ((A-tol < x[1]) && (x[1] < B+tol)))',
         '((near(x[1], A) || near(x[1], B)) && ((A-tol < x[0]) && (x[0] < B+tol)))']
CompiledSubDomain(' || '.join(f_str), A=A, B=B, tol=1E-10).mark(f, 1)
# Mesh to extend to (the multiplier mesh)
mesh_LM = EmbeddedMesh(f, 1)
```

```
---------------------------------------------------------------------------

ModuleNotFoundError                       Traceback (most recent call last)

<ipython-input-1-08ec97f2a11c> in <module>()
----> 1 from xii import EmbeddedMesh, Trace, Extension, ii_assemble, ii_convert, ii_Function
      2 from weak_bcs.la_solve import direct_solve_superlu
      3 from weak_bcs.bc_apply import apply_bc
      4 from weak_bcs.utils import block_form
      5 from dolfin import *


ModuleNotFoundError: No module named 'xii'
```

1.4 Variational form

In [4]:
```
V3d = FunctionSpace(mesh_3d, 'CG', 1)
V1d = FunctionSpace(mesh_1d, 'CG', 1)
Q = FunctionSpace(mesh_LM, 'CG', 1)

W = [V3d, V1d, Q]

u3d, u1d, p = map(TrialFunction, W)
v3d, v1d, q = map(TestFunction, W)

Tu3d, Tv3d = (Trace(f, mesh_LM) for f in (u3d, v3d))
Eu1d, Ev1d = (Extension(f, mesh_LM, type='uniform') for f in (u1d, v1d))

# Cell integral of Qspace
dxLM = Measure('dx', domain=mesh_LM)

# LHS
a = block_form(W, 2)

a[0][0] = inner(grad(u3d), grad(v3d))*dx
a[0][2] = -inner(Tv3d, p)*dxLM

a[1][1] = inner(grad(u1d), grad(v1d))*dx
a[1][2] = inner(Ev1d, p)*dxLM

a[2][0] = -inner(Tu3d, q)*dxLM
a[2][1] = inner(Eu1d, q)*dxLM

# RHS
L = block_form(W, 1)
L[1] = inner(Constant(1), v1d)*dx
```

1.5 Finishing up with boundary conditions and linear solve

In [5]:
```
A, b = map(ii_assemble, (a, L))

# Add boundary conditions
bcs = [[DirichletBC(V3d, Constant(0), 'on_boundary')],
       [DirichletBC(V1d, Expression('x[0]', degree=1), 'on_boundary')],
       [DirichletBC(Q, Constant(0), 'on_boundary')]]

A, b = apply_bc(A, b, bcs)

# Monolithic form solver
A, b = map(ii_convert, (A, b))

wh = direct_solve_superlu(A, b, W)
```

```
        for i, wi in enumerate(wh):
            File('extension_wh%d.pvd' % i) << wi
```

```
/home/mirok/.local/lib/python2.7/site-packages/scipy/sparse/linalg/dsolve/linsolve.py:296: Spars
  warn('splu requires CSC matrix format', SparseEfficiencyWarning)
```

```
('|b-Ax| from direct solver', 1.1571721078556396e-14)
```

1.6 Eccolo qua

```
In [11]: from IPython.display import Image
         from IPython.core.display import HTML
         Image(url="solution.png")
```

```
Out[11]: <IPython.core.display.Image object>
```