External operators in UFL and Firedrake

Nacime Bouziani David A. Ham

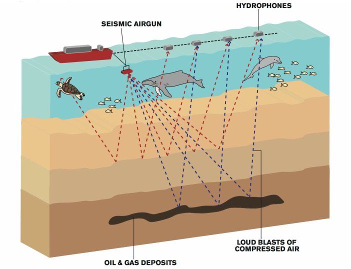
March 22-26, 2021

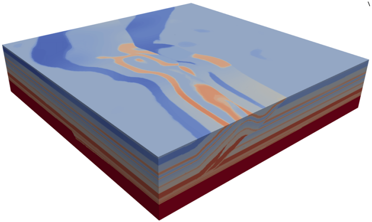
FEniCS 2021

Department of Mathematics, Imperial College London

Engineering and Physical Sciences Research Council

PDEs are not enough in many cases !





We often need terms not directly expressible in the vector calculus sense !

Implicit constitutive laws : Glen's flow law

Let's consider the following standard model for isothermal flow where we have to find $(u, p, \tau) \in V \times Q \times X$ with appropriate function spaces such that $\forall\ (w, \phi) \in V \times Q$ we have :

$$\begin{cases} \displaystyle\int \phi \nabla \cdot u = 0 & \textit{incompressibility} \\ \displaystyle\int -w \cdot \nabla p + (\nabla \cdot \tau_{i,j}) \cdot w - f \cdot w = 0 & \textit{stress balance} \\ \dfrac{1}{2}\left(\nabla u + \nabla u^T\right) = A|\tau|^2 \tau_{i,j} & \textit{Glen flow law} \end{cases} \tag{1}$$

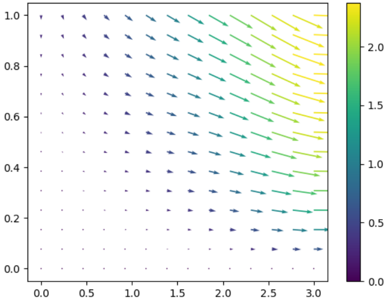
where $f = \begin{pmatrix} 0 \\ -\rho g \end{pmatrix}$ refers to the gravity force and $A \in \mathbb{R}$.

A Firedrake example: Pointwise solve operator

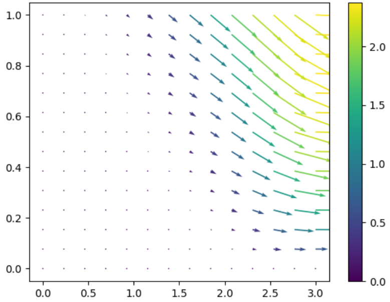
```python
import sympy as sp
...
# Define the function spaces
V1 = VectorFunctionSpace(mesh, "CG", 2)
V2 = FunctionSpace(mesh, "CG", 1)
V3 = TensorFunctionSpace(mesh, "DG", 2)
...
# Mixed function space
W = MixedFunctionSpace((V1, V2))
w, phi = TestFunctions(W)
soln = Function(W)
u, p = split(soln)
...
A = Constant(1.)
f = Function(V1).interpolate(as_vector([0, -rho*g]))
...

ps = point_solve(lambda tau,eps,A: A*sp.Matrix(tau)*sp.Matrix(tau).norm()**2 - eps,
                 function_space=V3,
                 solver_params={'x0':u0, 'maxiter':25, 'tol':1.e-7})
tau = ps(sym(grad(u)), A)
F = div(w)*p*dx - inner(grad(w), tau)*dx - phi*div(u)*dx - inner(f,w)*dx
...
# Solve
solve(F==0, soln, bcs=...)
```

Glen's flow law: velocity field



$$\tau = \mathcal{E}(u)$$



$$A|\tau|^2\tau_{i,j} = \mathcal{E}(u)$$

with $\mathcal{E}(u) = \frac{1}{2}\left(\nabla u + \nabla u^T\right)$

Framework

Let's consider the function space V and the parameter space M, which can be a function space or a subspace of $\mathbb{R}^m$ for $m \in \mathbb{N}$ depending on the applications. We introduce the so-called *external operator*

$$N : V \times M \to X \tag{2}$$

where X is the *external operator space*, it is a function space. N is external in the sense that **it can be defined externally with respect to Firedrake**.

Assembly

Let N be an ExternalOperator,

$$F(u, m, N(u, m); v) = 0 \quad \forall v \in V'$$

Assembly steps

$\ldots$

$u_h^X, m_h^X = \texttt{interpolate}(u_h, X), \texttt{interpolate}(m_h, X)$

$\ldots$

$\widehat{N} = N(u_h^X, m_h^X).\texttt{assemble}()$

$\ldots$

$\texttt{assemble}(F(u_h, m_h, \widehat{N}; v_h))$

Imperial College London

Assembly

Let N be an ExternalOperator,

$$F(u, m, N(u, m); v) = 0 \quad \forall v \in V'$$

Assembly steps

```
...
u_h^X , m_h^X = interpolate(u_h , X), interpolate(m_h , X)
...
N̂ = N(u_h^X, m_h^X).assemble()
...
assemble(F(u_h, m_h, N̂ ; v_h))
```

$\widehat{N}$ gets evaluated inside the operation of evaluating $F(u_h, m_h, \widehat{N}\,; v_h)$!

Differentiation rules

- Need to compute $\frac{\mathrm{d}F}{\mathrm{d}u}$, we have: $\frac{\mathrm{d}F}{\mathrm{d}u} = \frac{\partial F}{\partial u} + \frac{\partial F}{\partial N}\frac{\partial N}{\partial u}$
- Need to extend UFL to handle : $\frac{\partial F}{\partial N}$ and $\frac{\partial N}{\partial u}$
- The external operator subclass is responsible for computing $\frac{\partial N}{\partial u}$: SymPy, UFL, PyTorch...

Differentiation rules

- Need to compute $\frac{\mathrm{d}F}{\mathrm{d}u}$, we have: $\frac{\mathrm{d}F}{\mathrm{d}u} = \frac{\partial F}{\partial u} + \frac{\partial F}{\partial N}\frac{\partial N}{\partial u}$
- Need to extend UFL to handle : $\frac{\partial F}{\partial N}$ and $\frac{\partial N}{\partial u}$
- The external operator subclass is responsible for computing $\frac{\partial N}{\partial u}$: SymPy, UFL, PyTorch...

$$
\begin{tikzcd}
F\left(u, N(u); v\right) \arrow[rrr, "\frac{d}{du}"] \arrow[d] & & & \frac{d\,F(u,N(u),N'(u;\hat{u});\hat{u},v)}{d\,u} \arrow[dl] \arrow[dr] & \\
N(u) \arrow[d] & & N(u) \arrow[d] & & N'(u;\hat{u}) \arrow[d] \\
\widehat{N} & & \widehat{N} & & \widehat{N'}
\end{tikzcd}
$$

New Firedrake subclasses of ExternalOperator :

1. *Pointwise solve operator* : An operator that handles pointwise nonlinear relationships. More precisely, the pointwise solve operator is applied on a given UFL expression and provides the root of the function(al) defined by this expression.

2. *Neural Network* : The neural network operator takes an input and returns the output of the associated neural network model.

3. *Layer potentials* : This single (resp. double) layer potential operator computes the single (resp. double) layer potential (see Nonlocal UFL's talk (B. Sepanski) Thursday - 15:00–16:30 GMT).

Imperial College London

Example: PDE-constrained optimization

Let N_1 and N_2 be two external operators,

$$\min_{m \in M} \quad J(u, m, N_1(u, m)) \tag{3}$$

$$\text{subject to } F(u, m, N_2(u, m); v) = 0 \quad \forall v \in V' \tag{4}$$

where $J : V \times M \to \mathbb{R}$ is the objective function, $m \in M$ the control variable, and $u \in V$ is the weak solution of the parametrised PDE.

$\Rightarrow$ **Key objective** : Compute $\frac{\mathrm{d}J}{\mathrm{d}m}$

Adjoint equation

Using chain rule we get

$$\frac{\mathrm{d}J}{\mathrm{d}m} = -\lambda^* \left(\frac{\partial F}{\partial m} + \mu_1^* \right) + \mu_2^* + \frac{\partial J}{\partial m} \tag{5}$$

λ^*, μ_1^* and μ_2^* are the **adjoint variables**, they are obtained by the following relations:

$$\begin{cases} \left(\frac{\partial F}{\partial u} + \frac{\partial F}{\partial N_2} \frac{\partial N_2}{\partial u} \right)^* \lambda = \frac{\partial J}{\partial u}^* + \frac{\partial N_1}{\partial u}^* \frac{\partial J}{\partial N_1}^* \\ \mu_1 = \frac{\partial N_2}{\partial m}^* \frac{\partial F}{\partial N_2}^*, \ \mu_2 = \frac{\partial N_1}{\partial m}^* \frac{\partial J}{\partial N_1}^* \end{cases} \tag{6}$$

$\Rightarrow$ Adjoint computation depends on $\frac{\partial N_i}{\partial u}^*$ and $\frac{\partial N_i}{\partial m}^*$ for $i = 1, 2$

Imperial College London

Conclusion

In a nutshell:

1. The ExternalOperator project enables you to include any operators provided that you can define how the operator and its derivatives are evaluated. That can be anything that can be evaluated (e.g. Gaussian process, FFT, external libraries...)
2. Some classes of operator have already been implemented: **PointsolveOperator**, **PytorchOperator**, **SingleLayerPotential** and **DoubleLayerPotential**.
3. External operators play well with Pyadjoint, i.e. you can add in these operators in a PDE or PDE-constrained optimisation problem.
4. For neural networks, coupling with PyTorch to get derivative with respect to inputs/weights. Extensions to Tensorflow are straightforward.
5. External operators play well with matrix free methods.

Conclusion

In a nutshell:

1. The ExternalOperator project enables you to include any operators provided that you can define how the operator and its derivatives are evaluated. That can be anything that can be evaluated (e.g. Gaussian process, FFT, external libraries...)
2. Some classes of operator have already been implemented: **PointsolveOperator**, **PytorchOperator**, **SingleLayerPotential** and **DoubleLayerPotential**.
3. External operators play well with Pyadjoint, i.e. you can add in these operators in a PDE or PDE-constrained optimisation problem.
4. For neural networks, coupling with PyTorch to get derivative with respect to inputs/weights. Extensions to Tensorflow are straightforward.
5. External operators play well with matrix free methods.

What are the **practical takeaways** ?

Conclusion

In a nutshell:

1. The ExternalOperator project enables you to include any operators provided that you can define how the operator and its derivatives are evaluated. That can be anything that can be evaluated (e.g. Gaussian process, FFT, external libraries...)
2. Some classes of operator have already been implemented: **PointsolveOperator**, **PytorchOperator**, **SingleLayerPotential** and **DoubleLayerPotential**.
3. External operators play well with Pyadjoint, i.e. you can add in these operators in a PDE or PDE-constrained optimisation problem.
4. For neural networks, coupling with PyTorch to get derivative with respect to inputs/weights. Extensions to Tensorflow are straightforward.
5. External operators play well with matrix free methods.

What are the **practical takeaways** ?

- To build your own external operator: subclass the **AbstractExternalOperator** class in firedrake and equip your operator with an evaluate method (i.e. how your operator and its derivatives are evaluated).
- Code accessible via the **pointwise-adjoint-operator** firedrake branch
- Related talks:
  - External operators depend on dual spaces (see I. Marsden talk: Tuesday - 13:00 — 14:40 GMT).
  - LayerPotential operators (see B. Sepanski talk: Thursday - 15:00 — 16:30 GMT).