Firedrake

Explicit Dual Space Representation in UFL

India Marsden[1], David A. Ham[2] and Reuben Nixon-Hill[2,3]

March 2021

[1]Department of Computing, Imperial College London
[2]Department of Mathematics, Imperial College London
[3]Science and Solutions for a Changing Planet DTP, Grantham Institute for Climate Change and the Environment, Imperial College London

Imperial College London

Context

UFL provides an intuitive way to represent mathematical forms in code.
In particular, it is able to represent function spaces, finite elements within function spaces and functions on these spaces, among other things.

Problem

Typically, operations such as assemble are applied to the defined forms in UFL. Doing this results in objects that are not within UFL.

This means that the language is not *closed*.

```
element = FiniteElement("Lagrange", triangle, 1)
u = TrialFunction(element)
v = TestFunction(element)
f = Coefficient(element)

a = (u*v - inner(grad(u),grad(v)) ) * dx
L = f * v * dx
res = assemble(a)
res2 = assemble(L)
```

Examples of the problem

Operator Composition Where $\tau(u)$ is an external operator:

$$\text{grad}(u) \cdot \tau(u) \cdot \text{grad}(v) * dx$$

Interpolation Interpolation is not first class

$$\text{interp}(e, u) * v * dx$$

Adjoint Forward Operations

$$\text{action}(\text{interp}^*(\hat{e}, u), \text{adjoint}(u * v * dx))$$

Composing Assembled forms

$$\text{assemble}(v * dx + \text{assemble}(e * dx))$$

Dual Space

These operations depend on objects in the *dual* to the function space, the space of bounded linear functionals on V:

$$V^* = V \to \mathbb{R}$$

An example of an operation on a dual space is the Dirac Delta functional ($V^* \to \mathbb{R}$), ie point evaluation:

$$\delta_x(v) = v(x)$$

Dual Basis

A function space can be represented by its (primal) basis. A function in the space is then a set of coefficients of that basis:

$$v = v_i \phi_i \in V$$

A dual space can be similarly represented by *dual basis* functions, $\phi^* \in V \to \mathbb{R}$. Call the set of coefficients of a dual space a *cofunction*:

$$u = u_i \phi_i^* \in V^*$$

Writing $u(v)$ would be evaluation of the dual basis and result in a scalar.

1-Forms

In UFL, a 1-form represents a mathematical object with one unknown,such as below, which we can write in terms of the basis:

$$
\begin{aligned}
h(v) = \int_\Omega v \, dx &= \int_\Omega \phi_i \, dx \, v_i \\
&= \int_\Omega \phi_i \, dx \, I_{ij} v_j = \int_\Omega \phi_i \, dx \, \phi_i^*(\phi_j) v_j \\
&= \int_\Omega \phi_i \, dx \, \phi_i^*(v_j \phi_j) \\
&= \int_\Omega \phi_i \, dx \, \phi_i^*(v)
\end{aligned}
$$

Using the property $\phi_i^*(\phi_j) = \delta_{ij}$ and the linearity of the dual basis.

1-Forms

Therefore, we can see that 1-forms can be represented as cofunctions with coefficients:

$$h_i = \int_\Omega \phi_i \, dx$$

$$h = h_i \phi^*$$

This is a cofunction, an object in the dual space of V. Computationally, we write:

```
L = v * dx
obj = assemble(L)
```

Obviously, ***obj*** is not a current UFL object.

Interpolation

Define interpolation from a space U to a space V as the operator:

$$interp(u, v^*) : U \to V$$

We can write this as a form:

$$U \times V^* \to \mathbb{R}$$

As $V = V^{**} = V^* \to \mathbb{R}$. Then, taking the adjoint of this form we get:

$$V^* \times U \to \mathbb{R} = V^* \to U^*$$

which matches the expectation of linear operators.

Interpolation

Seeing interpolation as a function, we have the first argument as $u \in U$ and the second $v^* \in V$. Interpolation is dual evaluation of v^*:

$$\text{interp}(u, v^*) = v^*(u)$$

v^* is termed a *coargument*, and in code would be:

```
v_star = TestFunction(V.dual())
```

Introducing Cofunctions makes the adjoint behave correctly.

Draft Additions

With these draft additions, users will be able to write code such as:

```
V = FunctionSpace(domain, element)
v = TestFunction(V)
V_dual = V.dual()
L = v * dx
obj = assemble(L)
a = Cofunction(V)
res = a + obj
```

where res would be a valid operation and V_dual is the function space that is dual to V.

Further Implications

This change will need to be propagated into the implementation of *`assemble`* and other similar operations. This includes attaching data to these objects and adapting the implementations to take into account pre-assembled sections.