Proceedings of FEniCS 2021

22–26 March 2021

Editors

Igor Baratta

Jørgen S. Dokken

Chris Richardson

Matthew W. Scroggs

Explicit dual space representation in UFL

India Marsden, Department of Computing, Imperial College London, United Kingdom
David A Ham, Department of Mathematics, Imperial College London, United Kingdom
Reuben Nixon-Hill, Department of Mathematics, Imperial College London, United Kingdom

23 March 2021

This talk will discuss proposed changes to the Unified Form Language to include symbolic types representing dual spaces along with associated objects and functions. UFL represents forms over finite element spaces, and operations on these forms naturally results in objects in the dual space, or operators mapping to or from dual spaces. Since UFL currently does not have a representation of these objects, the language is not closed, meaning these operations result in objects outside of the language, which these changes aim to solve.

We will discuss the changes being made and their structure, the mathematical background and the potential benefits, applications and simplifications that this work enables.

This talk was awarded a prize: Best talk by a PhD student or undergraduate (runner up).

You can cite this talk as:

India Marsden, David A Ham, and Reuben Nixon-Hill. "Explicit dual space representation in UFL". In: *Proceedings of FEniCS 2021, online, 22–26 March* (eds Igor Baratta, Jørgen S. Dokken, Chris Richardson, Matthew W. Scroggs) (2021), 138–150. DOI: 10.6084/m9.figshare.14495250.

BibTeX for this citation can be found at https://mscroggs.github.io/fenics2021/talks/marsden.html.

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

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.

Imperial College London

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)
```

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

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

Interpolation Interpolation is not first class

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

Adjoint Forward Operations

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

Composing Assembled forms

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

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.

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:

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

We can write this as a form:

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

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

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

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.