Languages and Compilers for Variational Forms

Rob Kirby,* Matt Knepley† and L.R. Scott‡

October 23, 2010

Abstract

We discuss a language for variational forms that provides a way to specify the spaces used to define boundary conditions in the finite element method. This is linked with a geometry language that allows natural definitions of projections onto subspaces that incorporate boundary conditions.

Keywords: finite element, compiler, variational form

AMS Classification: 65D05, 65N15, 65N30

This report attempts to provide guidance regarding the development of a form compiler for finite element variational approximation [1]. We focus on simplicial elements since the corresponding technology for quadrilateral and hexahedral elements is better developed, under the name of spectral element methods. We tend to focus on simple Lagrange elements to understand whether higher degree elements, with larger numbers of edge-based and internal nodes, require different compilation strategies. Specific elements might require specific analysis of the sort presented here.

A primary objective here is to suggest a language for variational forms that includes the spaces used to specify boundary conditions. In a companion report [3] we examine different strategies for code generation for variational forms and finite elements. The notation used here is essentially the same as what was implemented in Analysa [2]. In particular, the use of spaces as variables in forms to define operators was a key feature in Analysa. Analysa also provided an elementary geometry language of domains to support projection operators for spaces, as is described briefly in Section 2.2.

*Department of Mathematics, Texas Tech University, Lubbock, Texas

†The Computation Institute, University of Chicago, Chicago, Illinois

‡The Computation Institute and Departments of Computer Science and Mathematics, University of Chicago, Chicago, Illinois

1 A language for variational forms

We propose a simple language for variational forms. In general, we consider multi-linear forms of any arity. Tri-linear forms are common, and ones of arity four arise.

The forms often have characteristic dimensions for which they are defined, both in terms of the spatial dimension of the space over which we integrate as well as the dimension of the image space of the (vector) functions being integrated. We introduce an operator $\mathcal{D}$ which identifies the appropriate dimensions. So for example, $\mathcal{D}(x) = 2$ means that the integration variable x is in a two-dimensional domain. Similarly, $\mathcal{D}(u) = 3$ would mean that the values of u are 3-dimensional vectors. It should be clear from the context which are the functions being integrated and which are the variables of integration (which appear after the "d" symbol at the end of the integral).

Let us list some forms that we could like to include. We begin with forms for scalar valued functions ($\mathcal{D}(u) = \mathcal{D}(v) = \mathcal{D}(w) = 1$) in any number of dimensions $\mathcal{D}(x) \geq 1$):

$$a_{\text{Laplace}}(u, v) = \int_\Omega (\nabla u) \cdot (\nabla v)\, dx \tag{1}$$

$$a_{\text{weightedLaplace}}(u, v, w) = \int_\Omega w(\nabla u) \cdot (\nabla v)\, dx \tag{2}$$

$$a_{\text{novecadvect}}(u, v) = \int_\Omega (u_{,j})v\, dx, \qquad j = 1, \ldots, \mathcal{D}(x) \tag{3}$$

where we use the notation $u_{,x}$ to denote the partial derivative of u in the x direction. Sometimes the dimension of the space and the dimension of (one of) the functions are linked:

$$a_{\text{advection}}(u, v, \mathbf{w}) = \int_\Omega (\mathbf{w} \cdot \nabla u)v\, dx, \quad \mathcal{D}(u) = \mathcal{D}(v) = 1,\ \mathcal{D}(\mathbf{w}) = \mathcal{D}(x) \geq 1 \tag{4}$$

$$a_{\text{Stokesgrad}}(\mathbf{u}, \mathbf{v}) = \int_\Omega (\nabla \mathbf{u}) : (\nabla \mathbf{v})\, dx, \quad \mathcal{D}(\mathbf{u}) = \mathcal{D}(\mathbf{v}) = \mathcal{D}(x) \geq 1 \tag{5}$$

$$a_{\text{divergence}}(\mathbf{u}, \mathbf{v}) = \int_\Omega (\nabla \cdot \mathbf{u})(\nabla \cdot \mathbf{v})\, dx, \quad \mathcal{D}(\mathbf{u}) = \mathcal{D}(\mathbf{v}) = \mathcal{D}(x) \geq 1 \tag{6}$$

$$b_{\text{pressure}}(\mathbf{u}, p) = \int_\Omega (\nabla \cdot \mathbf{u})p\, dx, \quad \mathcal{D}(\mathbf{u}) = \mathcal{D}(x) \geq 1,\ \mathcal{D}(p) = 1 \tag{7}$$

$$b_{\text{divtrans}}(\mathbf{u}, p) = -\int_\Omega \mathbf{u} \cdot \nabla p\, dx, \quad \mathcal{D}(\mathbf{u}) = \mathcal{D}(x) \geq 1,\ \mathcal{D}(p) = 1 \tag{8}$$

$$c_{\text{momentum}}(\mathbf{u}, \mathbf{v}, \mathbf{w}) = \int_\Omega (\mathbf{u} \cdot \nabla \mathbf{v}) \cdot \mathbf{w}\, dx, \quad \mathcal{D}(\mathbf{u}) = \mathcal{D}(\mathbf{v}) = \mathcal{D}(\mathbf{w}) = \mathcal{D}(x) \geq 1 \tag{9}$$

$$c_{\text{transport}}(u, \mathbf{v}, w) = \int_\Omega u(\mathbf{v} \cdot \nabla w)\, dx, \quad \mathcal{D}(u) = \mathcal{D}(w) = 1,\ \mathcal{D}(\mathbf{v}) = \mathcal{D}(x) \geq 1 \tag{10}$$

where we omit any multiplication operator multiplying a scalar times a tensor of rank one or more (vector, matrix, etc.). We use a dot "$\cdot$" to denote the vector dot product and a colon

":" to denote the Frobenius product of matrices. Some forms are special to a particular domain dimensions, such as

$$a_{\text{curl3D}}(\mathbf{u},\mathbf{v}) = \int_\Omega (\nabla\times \mathbf{u})\cdot(\nabla\times \mathbf{v})\,dx, \quad \mathcal{D}(\mathbf{u}) = \mathcal{D}(\mathbf{v}) = \mathcal{D}(x) = 3 \tag{11}$$

$$a_{\text{curl2D}}(\mathbf{u},\mathbf{v}) = \int_\Omega (\operatorname{curl}\mathbf{u})(\operatorname{curl}\mathbf{v})\,dx, \quad \mathcal{D}(\mathbf{u}) = \mathcal{D}(\mathbf{v}) = \mathcal{D}(x) = 2 \tag{12}$$

where $\nabla\times$ is the familiar "curl" operator from calculus in three dimensions, where as we use "curl" for the two dimensional case ($\operatorname{curl}\mathbf{u} = (u_2, 1-u_1, 2)$). Note that $\mathcal{D}(\nabla\times\mathbf{u}) = 3$ whereas $\mathcal{D}(\operatorname{curl}\mathbf{u}) = 1$.

Most of the differential operators in the above forms are well known from calculus. However, the strain tensor $\epsilon(\mathbf{u})$ may be less familiar:

$$\epsilon(\mathbf{u})_{ij} := \tfrac{1}{2}(u_{i,j} + u_{j,i}) = \tfrac{1}{2}(\nabla\mathbf{u} + \nabla\mathbf{u}^t)_{ij} \tag{13}$$

where we have used the transpose operator on the matrix $\nabla\mathbf{u}$ in the last term. Thus we have

$$a_{\text{strain}}(\mathbf{u},\mathbf{v}) = \int_\Omega \epsilon(\mathbf{u}) : \epsilon(\mathbf{v})\,dx, \quad \mathcal{D}(\mathbf{u}) = \mathcal{D}(\mathbf{v}) = \mathcal{D}(x) \geq 1 \tag{14}$$

In the forms described so far, functions are either vector or scalar valued. However, certain matrices arise through differentiation, so in general we might want to allow forms like

$$a_{\text{matrixLaplace}}(u, v, A) = \int_\Omega (A\nabla u)\cdot(\nabla v)\,dx \tag{15}$$

where A is a matrix, and we have again omitted any notation for matrix times vector multiplication.

Some terms in variational forms involve differentiation and others do not. Thus we can think of the variational form as a list of terms of the form

- a function, or
- a derivative of a function.

We are restricting to the case where the derivative is always a first-order differential operator of some sort (with constant coefficients). However, they can have an algebraic structure in that they map functions of one dimension to another. That is, `D` is defined for `u` of a given $\mathcal{D}$(`u`), and $\mathcal{D}$(`Du`) can be different.

But in addition, there is some form of multiplication of terms that depends on the type (dimension) of the values of the functions. If we let `m` denote a typical multiplication operator and `D` a typical differential operator, then we can represent all of the forms considered so far as the integral over the domain Ω of an expression of the form

$$(\mathtt{u}\ \mathtt{m}_1(\mathtt{D}_1\mathtt{v})\ \mathtt{m}_2(\mathtt{D}_2\mathtt{w})) \tag{16}$$

We could simplify the notation by including the trivial differential operator: `Du = u`. Of course, the dimensions of the various terms have got to match.

We can now summarize some of the things that will be needed in compiling forms of this type.

- the algebraic form of the differential operators `D` will obviously play a role (consider ∇, $\nabla\cdot$, $\nabla\times$, etc.)
- The "type" of the multiplication operators `m`, and the variable `A`, `u`, `v`, `w`, etc.
- Possible symmetries of the form.

1.1 More complicated variational forms

Some variational forms have a more complicated description. This can occur in different ways. The simplest case is when there are multiple forms added together. This will be addressed in subsequent subsections (see Section 3.2) where we give examples of forms from different applications. Another case might occur when a form is the derivative of a form with respect to one of the functions, as occurs in Newton's method. Finally, more complicated combinations of functions and derivatives may appear, or simple coefficients may need to be specified. We turn now to this situation.

1.2 Defining complicated coefficients

Consider

$$a_{\text{nonlinearLaplace}}(u,v,w) = \int_\Omega f(\nabla w)\nabla u \cdot \nabla v \, dx \tag{17}$$

where f is some function of d variables (d is the dimension of Ω) with positive real values. A common expression might be $f(\xi) := 1/\sqrt{1+|\xi|^2}$.

This does not easily fit the simple model. However, often such nonlinear coefficients do not directly enter into a problem. Rather, the function w is fixed and the resulting bilinear form is used. We can force (17) into the simple model by creating an appropriate function F which represents $f(\nabla w)$. The we write

$$a_{\text{newnonlinearLaplace}}(u,v,w) = \int_\Omega F \cdot (\nabla u \cdot \nabla v) \, dx \tag{18}$$

where f is some function of d variables (d is the dimension of Ω) with positive real values. Note that we included a $\cdot$ between F and $(\nabla u \cdot \nabla v)$ to avoid ambiguity. It does not seem restrictive to require that all multiplications be specified completely.

We need to say at some point what class of functions we are considering in our variational forms. If they do not have a finite representation, then it will not be simple to evaluate the integrals. One approach to take is that all integrals will be computed by numerical quadrature involving simple point values. This of course requires us to be able to tabulate the derivatives of functions in the spaces we use at quadrature points as well as the function values.

If we take this approach, all that is needed in creating a function F from an expression such as $f(\nabla w)$ is the ability to evaluate the latter expression at quadrature points. That is, we assume that we have some operator which is defined on expressions involving function values and derivatives and can produce the corresponding values at quadrature points.

Let us be more precise by introducing some notation. We will assume that our variational forms are defined on $u \in \mathcal{U}$, $v \in \mathcal{V}$ and $w \in \mathcal{W}$ for finite dimensional function spaces $\mathcal{U}$, $\mathcal{V}$ and $\mathcal{W}$. Of course we might have $\mathcal{U} = \mathcal{V} = \mathcal{W}$ in some cases. Typical examples for these spaces would be piecewise polynomials, trigonometric functions, wavelets, and so forth.

We assume that, for all the spaces used for a given form, there is a fixed quadrature rule that can be used for the appropriate products of function values and derivatives. Moreover, we assume that there are mappings such as

$$\mathcal{U} \to \mathcal{Q} \tag{19}$$

which take the spaces to the corresponding set of quadrature points. Finally, to deal with complex expressions, we assume that there is a function that maps

$$E \times \mathcal{U} \to \mathcal{Q}, \tag{20}$$

where E stands for the set of expressions allowed in our language. We postpone definition of such expressions for the moment, but let us assume something equivalent to what is in Scheme is available.

2 The Action of Forms

Consider a variational problem to find $u \in \mathcal{V}$ such that

$$a(v, u) = F(v) \quad \forall v \in \mathcal{V} \tag{21}$$

for a given variational (continuous, bilinear) form $a(\cdot, \cdot)$. In general, we will want to consider more general forms (defined on pairs of different spaces, etc., but for now we will keep it simple.) This corresponds to a linear system of equations and one can write the corresponding matrix equation in terms of a basis. In practice, an interpolation basis is typically used, e.g., the standard Lagrange nodal basis $\{\phi_i \ : \ i \in \mathcal{I}\}$ where $\mathcal{I}$ denotes the index set for the nodes. The matrix equation is

$$AU = F \quad \forall v \in \mathcal{V} \tag{22}$$

where

$$\begin{aligned} A_{ij} &:= a(\phi_i, \phi_j) \\ F_j &:= F(\phi_j) \\ u &:= \sum_{i \in \mathcal{I}} U_i \phi_i \end{aligned} \tag{23}$$

In many iterative methods, the actual matrix A is not needed explicitly, rather all that is required is some way to compute the **action** of A, that is, the mapping that sends a vector V to the vector AV. This operation can be defined purely in terms of the bilinear form as follows. Suppose we write

$$v := \sum_{i \in \mathcal{I}} V_i \phi_i \tag{24}$$

Then for all $i \in \mathcal{I}$

$$\begin{aligned}(AV)_i =& \sum_{j\in\mathcal{I}} A_{ij}V_j \\ =& \sum_{j\in\mathcal{I}} a(\phi_i,\phi_j)V_j \\ =& a(\phi_i, \sum_{j\in\mathcal{I}} V_j\phi_j) \\ =& a(\phi_i, v)\end{aligned} \tag{25}$$

Thus the vector AV can be computed by evaluating $a(\phi_i, v)$ for all $i \in \mathcal{I}$. It may not be clear how this can be done efficiently, but it turns out that the standard matrix assembly algorithm can be used to compute the action efficiently.

2.1 Action Notation

Having notation for variational forms allows us to generate code for a wide variety of differential equations. However, there is also a need to specify the associated operators, such as the action. There is no generic action that can be derived from a form; instead one has to specify the space upon which it acts. With (25) as motivation, we can introduce the notation [2]

$$\boxed{AV = a(\mathcal{V}, v)} \tag{26}$$

where the "$\mathcal{V}$" indicates implicitly the range of the index variable i. Note that evaluating $Y_i := a(v, \phi_i)$ for all $i \in \mathcal{I}$ computes the vector $Y = A^tV$. In the notation of (26), we have $A^tV = a(v, \mathcal{V})$.

The action of a bilinear form can be used in several contexts. Perhaps the simplest is when non-homogeneous boundary conditions are posed. Suppose g represents a function defined on the whole domain which satisfies the correct boundary conditions. A typical variational problem is to find u such that $u - g \in \mathcal{V}$ and

$$a(v, u) = 0 \quad \forall v \in \mathcal{V}. \tag{27}$$

This can be re-written using the difference $u^0 := u - g \in \mathcal{V}$. The variational problem becomes: Find $u^0 \in \mathcal{V}$ such that

$$a(v, u^0) = -a(v, g) \quad \forall v \in \mathcal{V}. \tag{28}$$

In matrix form, we would write this as

$$AU^0 = -a(\mathcal{V}, g). \tag{29}$$

This could be solved by a direct method (e.g., Gaussian elimination) with $-a(\mathcal{V}, g)$ as the right-hand-side vector.

2.2 A geometry language

Although the function g could in principle come from any space, in practice it is easier if it comes from a related finite element space. For example, with $\mathcal{V}$ representing homogeneous Dirichlet boundary conditions, we can think of $\mathcal{V}$ as a subspace of a larger space $\widetilde{\mathcal{V}}$ made from the same finite elements, but without any restrictions at the boundary. If we start with $g \in \widetilde{\mathcal{V}}$, then the resulting u is also in $\widetilde{\mathcal{V}}$. In Analysa [2], projection operators were provided to define $\mathcal{V} \subset \widetilde{\mathcal{V}}$ simply by indicating the subdomain on which they were supported. This required interaction with the underlying geometry system and specifications of a language for domains, including operations such as "interior" and "boundary." In particular, finite element spaces were always defined with reference to a particular domain. Typical usage involved defining the largest space, e.g., $\widetilde{\mathcal{V}}$, and then defining a smaller space such as $\mathcal{V}$ via a projection onto the interior. Projections were defined in Analysa [2] for both functions and operators in a compatible way. For example, it is easy to see that the solution u is independent of the values of g in the interior. Thus it is only necessary to define g on a boundary space derived from the projection of $\widetilde{\mathcal{V}}$ onto the boundary.

Analysa [2] provided an intrinsic operator `fe` to construct finite element spaces from two ingredients, an element and a mesh. Typical syntax was

```
(fe an-element (all a-mesh) r^2:)
(fe an-element (boundary a-mesh) r^2:)
(fe an-element ((- all boundary) a-mesh) r^2:)
```

where the third variable indicated that this space was in two dimensions (Analysa supported only two- and three-dimensional problems). Here the element `an-element` had to be previously defined, as well as the mesh `a-mesh`. The built-in operators `all` and `boundary` corresponded to the closure of the domain and the boundary of the domain respectively. The interior of the domain was determined by the calculus of set minus:

```
(- all boundary)
```

(which means `all` minus `boundary`). All three of the above definitions produced different spaces $\mathcal{V}$ that were frequently used.

In addition to the `fe` operator used to construct spaces, there was also a built-in operator in Analysa [2] to project between related spaces. For example, suppose that we give the names `Vee-all`, `Vee-boundary`, and `Vee-interior`, respectively, to the above spaces. Then for a member `you` in `Vee-all`, the expression

```
(projection you Vee-interior)
```

would correctly produce a member of `Vee-interior`. The space `Vee-all` corresponds to $\widetilde{\mathcal{V}}$ in the previous notation, and `Vee-interior` corresponds to $\mathcal{V}$. The space `Vee-boundary` corresponds to a new space, call it $\mathcal{V}_\partial$, with the property that

$$\widetilde{\mathcal{V}} = \mathcal{V} \oplus \mathcal{V}_\partial.$$

3 Multi-linear forms and different spaces

The nonlinear term in the Navier–Stokes provides an example of the action of a general multi-linear form. Recall that it is of the form

$$\begin{aligned} c(\mathbf{u},\mathbf{v},\mathbf{w}) &:= \int \mathbf{u}(x)\cdot\nabla\mathbf{v}(x)\cdot\mathbf{w}(x)\,dx \\ &= \int \sum_{j,k=1}^{d} u_j(x)\frac{\partial}{\partial x_j}v_k(x)w_k(x)\,dx \end{aligned} \tag{30}$$

The special case $\mathbf{u}=\mathbf{v}$ occurs frequently in many algorithms. Suppose there is a variational equation of the form to find $\mathbf{u}\in\mathcal{V}$ such that

$$a(\mathbf{u},\mathbf{w}) = c(\mathbf{v},\mathbf{v},\mathbf{w}) \quad \forall \mathbf{w}\in\mathcal{V} \tag{31}$$

for some given $\mathbf{v}\in\mathcal{V}$. Choose $\mathbf{w}=\phi_i$ for a generic basis function ϕ_i. Write as usual $\mathbf{u}:=\sum_{i\in\mathcal{I}}U_i\phi_i$. Then

$$\begin{aligned} \left(A^tU\right)_i =& \sum_{j\in\mathcal{I}} A_{ji}U_j \\ =& \sum_{j\in\mathcal{I}} a(\phi_j,\phi_i)U_j \\ =& a(\sum_{j\in\mathcal{I}} U_j\phi_j,\phi_i) \\ =& a(\mathbf{u},\phi_i) \\ =& c(\mathbf{v},\mathbf{v},\phi_i) \quad \forall i\in\mathcal{I}. \end{aligned} \tag{32}$$

In notation analogous to that of (26), we can write (32) as

$$A^tU = c(\mathbf{v},\mathbf{v},\mathcal{V}). \tag{33}$$

Now suppose that instead the variational equation is to find $\mathbf{u}\in\mathcal{V}$ such that

$$a(\mathbf{u},\mathbf{w}) = c(\mathbf{v},\tilde{\mathbf{v}},\mathbf{w}) \quad \forall \mathbf{w}\in\mathcal{V} \tag{34}$$

for two different $\mathbf{v}\in\mathcal{V}$ and $\tilde{\mathbf{v}}\in\mathcal{V}$. Then (34) becomes $A^tU = c(\mathbf{v},\tilde{\mathbf{v}},\mathcal{V})$, with the obvious definition

$$\left(c(\mathbf{v},\tilde{\mathbf{v}},\mathcal{V})\right)_i := c(\mathbf{v},\tilde{\mathbf{v}},\phi_i) \quad \forall i\in\mathcal{I}. \tag{35}$$

With forms of two or more variables, there are other objects that can be generated automatically in a way that is similar to what we can do to generate the action of a form. In particular, we recall the definition of A in (23):

$$A_{ij} := a(\phi_i,\phi_j) \quad \forall i,j\in\mathcal{I} \tag{36}$$

and, by a simple extension of our convention (26), we can view this as equivalent to

$$A = a(\mathcal{V},\mathcal{V}). \tag{37}$$

For trivariate forms such as (30), it might be of interest to work with the matrix

$$C_{ij} := c(\mathbf{v}, \phi_i, \phi_j) \quad \forall i, j \in \mathcal{I} \tag{38}$$

which we write in our shorthand as

$$C = c(\mathbf{v}, \mathcal{V}, \mathcal{V}) \tag{39}$$

For example, one might want to solve (for $\mathbf{u}$, given $\mathbf{f}$) the equation

$$\mathbf{u} + \mathbf{v} \cdot \nabla \mathbf{u} = \mathbf{f} \tag{40}$$

for a fixed, specified $\mathbf{v} \in \mathcal{V}$, using the variational form

$$(\mathbf{u}, \mathbf{w}) + c(\mathbf{v}, \mathbf{u}, \mathbf{w}) = (\mathbf{f}, \mathbf{w}) \quad \forall \mathbf{w} \in \mathcal{V} \tag{41}$$

to define $\mathbf{u} \in \mathcal{V}$. In component form, this becomes

$$\sum_{i \in \mathcal{I}} U_i \left((\phi_i, \phi_j) + c(\mathbf{v}, \phi_i, \phi_j) \right) = (\mathbf{f}, \phi_j) \quad \forall j \in \mathcal{I} \tag{42}$$

In matrix notation, this becomes

$$U^t \left(M + c(\mathbf{v}, \mathcal{V}, \mathcal{V}) \right) = F \tag{43}$$

3.1 Using different spaces

It may frequently happen that the spaces in a form are not all the same. Consider forms and spaces such as in any of the following examples:

$$\begin{aligned} b(v, p) &:= \int \nabla \cdot \mathbf{v}(x) p(x) \, dx \\ c(w, \mathbf{v}, z) &:= \int w(x) \mathbf{v}(x) \cdot \nabla z(x) \, dx \end{aligned} \tag{44}$$

The form $b(\cdot, \cdot)$ involves spaces of scalar functions (say, Π) as well as vector functions (say, $\mathcal{V}$). The matrix $b(\mathcal{V}, \Pi)$ is defined analogously to (36) and (37):

$$\left(b(\mathcal{V}, \Pi) \right)_{ij} := b(\phi_i, q_i) \tag{45}$$

where $\{\phi_i \, : \, i \in \mathcal{I}\}$ is a basis of $\mathcal{V}$ as before, and $\{q_i \, : \, i \in \mathcal{J}\}$ is a basis of Π. Note that $b(\mathcal{V}, \Pi)$ will not, in general, be a square matrix.

In general, if we have a form $a(v^1, \ldots, v^n)$ of n entries, then the expression

$$a\left(\ldots, \mathcal{V}^1, \ldots, \mathcal{V}^k, \ldots \right) \tag{46}$$

defines a tensor of rank k. More precisely, each of the n arguments in the form $a(v^1, \ldots, v^n)$ may be a function space or a member of a function space. Suppose that we have some k of the n (not necessarily consecutive) arguments that are function spaces, $\mathcal{V}^1, \ldots, \mathcal{V}^k$,

and $n-k$ members of a function space $v^1, \ldots, v^{n-k}$. Then, suppose we have a partition of $\{1, \ldots, n\} = \{i_\ell \,:\, \ell = 1, \ldots k\} \bigcup \{j_\ell \,:\, \ell = 1, \ldots n-k\}$ suppose that $w_{i_\ell} \to \mathcal{V}^{i_\ell}$ for $\ell = 1, \ldots k$ and $w_{j_\ell} \to v^{j_\ell}$ for $\ell = 1, \ldots n-k$. Here $a \to b$ should be read as "a is a pointer to b." Then $a(w_1, \ldots, w_n)$ is a rank k tensor. For example, $a\left(v^1, v^2, \mathcal{V}^1, v^3, \mathcal{V}^2, \mathcal{V}^3, v^4\right)$ denotes a tensor of rank 3, whereas $a\left(v^1, \mathcal{V}^1, v^2, v^3, \mathcal{V}^2, v^4, v^5\right)$ denotes a tensor of rank 2.

Note that a tensor of rank zero is just a scalar, consistent with the usual interpretation of $a(v^1, \ldots, v^n)$. A tensor of rank one is a vector, and a tensor of rank two is a matrix. Tensors of rank three or higher are less common in computational linear algebra.

3.2 Streamline diffusion

Simulation of high-speed fluid-flows requires sophisticated methods. One method uses continuous piecewise linear functions $\mathcal{W}$ for pressure and velocity ($\mathcal{V} = \mathcal{W} \times \mathcal{W} \times \mathcal{W}$). The basic time-step (of size Δt) defines $(\mathbf{u}^n, p) \in \mathcal{V} \times \mathcal{W}$ from $\mathbf{u}^{n-1}$ via a variational formulation

$$\begin{aligned}&((\mathbf{u}^n - \mathbf{u}^{n-1})\Delta t^{-1}, \mathbf{v}) + c_{\text{momentum}}(\hat{\mathbf{u}}^n, \hat{\mathbf{u}}^n, \mathbf{v}) + a_{\text{nustrain}}(\hat{\mathbf{u}}^n, \mathbf{v}) - b_{\text{pressure}}(\mathbf{v}, p) \\ &+ b_{\text{pressure}}(\hat{\mathbf{u}}^n, q) + a_{SD}(\delta, \hat{\mathbf{u}}^n, p, \mathbf{v}, q) = (\mathbf{f}, \mathbf{v} + \delta_1(\hat{\mathbf{u}}^n)(\hat{\mathbf{u}}^n \cdot \nabla \mathbf{v} + \nabla q)) \quad \forall (\mathbf{v}, q) \in \mathcal{V} \times \mathcal{W},\end{aligned} \tag{47}$$

where the form $a_{\text{nustrain}}(\cdot, \cdot)$ is 2ν times the strain form defined in (14), the form $b_{\text{pressure}}(\cdot, \cdot)$ was defined in (7), the form $c_{\text{momentum}}(\cdot, \cdot, \cdot)$ was defined in (9), and $\hat{\mathbf{u}}^n = \frac{1}{2}(\mathbf{u}^n + \mathbf{u}^{n-1})$. The streamline diffusion form provides the stabilizing term and is defined by

$$\begin{aligned}a_{SD}(\delta(\tilde{\mathbf{u}}), \mathbf{u}, p, \mathbf{v}, q) :=& (\delta_1(\tilde{\mathbf{u}})(\mathbf{u} \cdot \nabla \mathbf{u} + \nabla p), \mathbf{u} \cdot \nabla \mathbf{v} + \nabla q) + (\delta_2(\tilde{\mathbf{u}}) \nabla \cdot \mathbf{u}, \nabla \cdot \mathbf{v}) \\ =& (\delta_1(\tilde{\mathbf{u}})(\mathbf{u} \cdot \nabla \mathbf{u}), \mathbf{u} \cdot \nabla \mathbf{v}) + (\delta_1(\tilde{\mathbf{u}})(\mathbf{u} \cdot \nabla \mathbf{u}), \nabla q) + (\delta_1(\tilde{\mathbf{u}}) \nabla p, \mathbf{u} \cdot \nabla \mathbf{v}) \\ &+ (\delta_1(\tilde{\mathbf{u}}) \nabla p, \nabla q) + (\delta_2(\tilde{\mathbf{u}})(\nabla \cdot \mathbf{u}), \nabla \cdot \mathbf{v}),\end{aligned} \tag{48}$$

and the functions $\delta_i = \delta_i(\tilde{\mathbf{u}})$ are piecewise constants defined as follows. Let U denote a mean value of $|\tilde{\mathbf{u}}|$ on a given element. Then on that element, $\delta_1 = \frac{1}{2}(k_n^{-2} + U^{-2})^{-1/2}$ in the convection-dominated case ($\nu < Uh$) and $\delta_1 = \kappa_1 h^2$ otherwise, $\delta_2 = \kappa_2 h$ if $\nu < Uh$ and $\delta_2 = \kappa_2 h^2$ otherwise, with κ_1 and κ_2 positive constants of unit size.

Let us define the various components of the form $a_{SD}(\cdot, \cdot)$ as follows:

$$\begin{aligned}a_{\text{SDadvec}}(\gamma, \mathbf{u}, \mathbf{v}, \mathbf{w}) &:= \int_\Omega \gamma(\mathbf{u} \cdot \nabla \mathbf{v}) \cdot (\mathbf{u} \cdot \nabla \mathbf{w})\, dx \\ a_{\text{SDpradv}}(\gamma, \mathbf{u}, \mathbf{v}, q) &:= \int_\Omega \gamma(\mathbf{u} \cdot \nabla \mathbf{v}) \cdot \nabla q\, dx \\ a_{\text{SDlaPress}}(\gamma, p, q) &:= \int_\Omega \gamma \nabla p \cdot \nabla q\, dx \\ a_{\text{SDdiv}}(\gamma, \mathbf{u}, \mathbf{v}) &:= \int_\Omega \gamma(\nabla \cdot \mathbf{u})(\nabla \cdot \mathbf{v})\, dx\end{aligned} \tag{49}$$

Thus we can write

$$\begin{aligned}a_{SD}(\delta(\tilde{\mathbf{u}}), \mathbf{u}, p, \mathbf{v}, q) =& a_{\text{SDadvec}}(\delta_1(\tilde{\mathbf{u}}), \mathbf{u}, \mathbf{u}, \mathbf{v}) + a_{\text{SDdiv}}(\delta_2(\tilde{\mathbf{u}}), \mathbf{u}, \mathbf{v}) \\ & a_{\text{SDpradv}}(\delta_1(\tilde{\mathbf{u}}), \mathbf{u}, \mathbf{u}, q) + a_{\text{SDpradv}}(\delta_1(\tilde{\mathbf{u}}), \mathbf{u}, \mathbf{v}, p) + \\ & a_{\text{SDlaPress}}(\delta_1(\tilde{\mathbf{u}}), p, q)\end{aligned} \tag{50}$$

We can write $\mathbf{u}^n - \mathbf{u}^{n-1} = 2(\hat{\mathbf{u}}^n - \mathbf{u}^{n-1})$, so that (47) can be written as an equation for $\hat{\mathbf{u}}^n$ as

$$\begin{aligned} & m_{\text{mass}}(\hat{\mathbf{u}}^n, \mathbf{v}) + c_{\text{momentum}}(\hat{\mathbf{u}}^n, \hat{\mathbf{u}}^n, \mathbf{v}) + a_{\text{nustrain}}(\hat{\mathbf{u}}^n, \mathbf{v}) \\ & - b_{\text{pressure}}(\mathbf{v}, p) + b_{\text{pressure}}(\hat{\mathbf{u}}^n, q) + a_{SD}(\delta(\hat{\mathbf{u}}^n), \hat{\mathbf{u}}^n, p, \mathbf{v}, q) \\ =& (\mathbf{f}, \mathbf{v} + \delta_1(\hat{\mathbf{u}}^n)(\hat{\mathbf{u}}^n \cdot \nabla \mathbf{v} + \nabla q)) + m_{\text{mass}}(\mathbf{u}^{n-1}, \mathbf{v}) \quad \forall (\mathbf{v}, q) \in \mathcal{V} \times \mathcal{W}, \end{aligned} \tag{51}$$

where $m_{\text{mass}}(\cdot, \cdot)$ is the L^2 inner-product times $2\Delta t^{-1}$. The equation (51) is nonlinear, so we need to use some algorithm to solve it. Using a simple fixed-point iteration leads to a sequence of linear problems of the following type: $\forall (\mathbf{v}, q) \in \mathcal{V} \times \mathcal{W}$,

$$\begin{aligned} & m_{\text{mass}}(\hat{\mathbf{u}}^{n,\ell}, \mathbf{v}) + c_{\text{momentum}}(\hat{\mathbf{u}}^{n,\ell-1}, \hat{\mathbf{u}}^{n,\ell}, \mathbf{v}) + a_{\text{nustrain}}(\hat{\mathbf{u}}^{n,\ell}, \mathbf{v}) \\ & \quad - b_{\text{pressure}}(\mathbf{v}, p) + b_{\text{pressure}}(\hat{\mathbf{u}}^{n,\ell}, q) \\ & \quad + a_{\text{SDadvec}}(\delta_1(\hat{\mathbf{u}}^{n,\ell-1}), \hat{\mathbf{u}}^{n,\ell-1}, \hat{\mathbf{u}}^{n,\ell}, \mathbf{v}) + a_{\text{SDdiv}}(\delta_2(\hat{\mathbf{u}}^{n,\ell-1}), \hat{\mathbf{u}}^{n,\ell}, \mathbf{v}) \\ & \quad + a_{\text{SDpradv}}(\delta_1(\hat{\mathbf{u}}^{n,\ell-1}), \hat{\mathbf{u}}^{n,\ell-1}, \hat{\mathbf{u}}^{n,\ell}, q) + a_{\text{SDpradv}}(\delta_1(\hat{\mathbf{u}}^{n,\ell-1}), \hat{\mathbf{u}}^{n,\ell-1}, \mathbf{v}, p) \\ & \quad + a_{\text{SDlaPress}}(\delta_1(\hat{\mathbf{u}}^{n,\ell-1}), p, q) \\ =& (\mathbf{f}, \mathbf{v} + \delta_1(\hat{\mathbf{u}}^n)(\hat{\mathbf{u}}^n \cdot \nabla \mathbf{v} + \nabla q)) + m_{\text{mass}}(\mathbf{u}^{n-1}, \mathbf{v}) \\ =& (\mathbf{f}, \mathbf{v}) + b_{\text{divtrans}}(\delta_1(\hat{\mathbf{u}}^{n,\ell-1})\mathbf{f}, q) \\ & \quad + c_{\text{momentum}}(\hat{\mathbf{u}}^{n,\ell-1}, \mathbf{v}, \delta_1(\hat{\mathbf{u}}^{n,\ell-1})\mathbf{f}) + m_{\text{mass}}(\mathbf{u}^{n-1}, \mathbf{v}) \end{aligned} \tag{52}$$

where we can start with $\hat{\mathbf{u}}^{n,0} = \mathbf{u}^{n-1}$ or some extrapolation of previous $\mathbf{u}$'s to the current time step, e.g., $\hat{\mathbf{u}}^{n,0} = \frac{3}{2}\mathbf{u}^{n-1} - \frac{1}{2}\mathbf{u}^{n-2}$. The form $b_{\text{divtrans}}(\cdot, \cdot)$ is defined in (8).

The variational formulation (52) leads to a system of the following matrix system:

$$\begin{aligned} A^\ell U^\ell + B^\ell P^\ell &= F, \\ \widetilde{B}^\ell U^\ell + C^\ell P^\ell &= G, \end{aligned} \tag{53}$$

where the matrices A^ℓ, B^ℓ, $\widetilde{B}^\ell$ and C^ℓ are defined by

$$\begin{aligned} A^\ell =& m_{\text{mass}}(\mathcal{V}, \mathcal{V}) + c_{\text{momentum}}(\hat{\mathbf{u}}^{n,\ell-1}, \mathcal{V}, \mathcal{V}) + a_{\text{nustrain}}(\mathcal{V}, \mathcal{V}) \\ & + a_{\text{SDadvec}}(\delta_1(\hat{\mathbf{u}}^{n,\ell-1}), \hat{\mathbf{u}}^{n,\ell-1}, \mathcal{V}, \mathcal{V}) + a_{\text{SDdiv}}(\delta_2(\hat{\mathbf{u}}^{n,\ell-1}), \mathcal{V}, \mathcal{V}) \\ B^\ell =& - b_{\text{pressure}}(\mathcal{V}, \mathcal{W}) + a_{\text{SDpradv}}(\delta_1(\hat{\mathbf{u}}^{n,\ell-1}), \hat{\mathbf{u}}^{n,\ell-1}, \mathcal{V}, \mathcal{W}) \\ (\widetilde{B}^\ell)^\top =& b_{\text{pressure}}(\mathcal{V}, \mathcal{W}) + a_{\text{SDpradv}}(\delta_1(\hat{\mathbf{u}}^{n,\ell-1}), \hat{\mathbf{u}}^{n,\ell-1}, \mathcal{V}, \mathcal{W}) \\ C^\ell =& a_{\text{SDlaPress}}(\delta_1(\hat{\mathbf{u}}^{n,\ell-1}), \mathcal{W}, \mathcal{W}) \end{aligned} \tag{54}$$

and the vectors on the right hand side are defined by

$$\begin{aligned} F &= (\mathbf{f}, \mathcal{V}) + c_{\text{momentum}}(\hat{\mathbf{u}}^{n,\ell-1}, \mathcal{V}, \delta_1(\hat{\mathbf{u}}^{n,\ell-1})\mathbf{f}) + m_{\text{mass}}(\mathbf{u}^{n-1}, \mathcal{V}) \\ G &= b_{\text{divtrans}}(\delta_1(\hat{\mathbf{u}}^{n,\ell-1})\mathbf{f}, \mathcal{W}) \end{aligned} \tag{55}$$

This system can be solved using another fixed point (inner) iteration, where we first solve for $P^{n,j+1}$ in terms of $U_h^{n,j}$ from the equation

$$C^\ell P^{\ell,j} = G - \widetilde{B}^\ell U_h^{\ell,j} \tag{56}$$

(e.g., using a multigrid method), and then solve for $U_h^{\ell,j+1}$ from the equation

$$A^\ell U_h^{\ell,j+1} = F - B^\ell P^{\ell,j} \tag{57}$$

using GMRES.

4 Evaluation of Bilinear Forms for Isoparametrics Elements

When using high-order elements for problems with curved boundaries, it is essential to include some way of approximating the boundary conditions accurately. One of the most effective in engineering practice is to use isoparametrics elements. In this approach, the element basis functions ϕ_λ^e are related to the reference basis functions via a polynomial mapping, $\xi \to F(\xi)$, of $\mathcal{T}$ to T_e:

$$\phi_\lambda^e(x) = \phi_\lambda\left(F^{-1}(x)\right).$$

The use of element matrices becomes quite complicated with isoparametrics elements. Let us return to the example involving $a(\cdot,\cdot)$. We have

$$\begin{aligned} a_e(\mathbf{v},\mathbf{w}) = &\int_{\mathcal{T}} \sum_{j,k=1}^{d} \left(J_F^{-1}(\xi)^t \sum_{\lambda\in\mathcal{L}} v_k^{\iota(e,\lambda)} \nabla\phi_\lambda(\xi) \right)_j \times \\ &\left(J_F^{-1}(\xi)^t \sum_{\lambda\in\mathcal{L}} w_k^{\iota(e,\lambda)} \nabla\phi_\lambda(\xi) \right)_j \det(J_F(\xi))\, d\xi \\ = &\sum_{\xi\in\Xi} \omega_\xi \sum_{j,k=1}^{d} \left(J_F^{-1}(\xi)^t \sum_{\lambda\in\mathcal{L}} v_k^{\iota(e,\lambda)} \nabla\phi_\lambda(\xi) \right)_j \times \\ &\left(J_F^{-1}(\xi)^t \sum_{\lambda\in\mathcal{L}} w_k^{\iota(e,\lambda)} \nabla\phi_\lambda(\xi) \right)_j \det(J_F(\xi)) \,. \end{aligned} \tag{58}$$

Writing this in terms of element matrices is quite similar to using trilinear forms, i.e., $a_e(u,v) = a_e(F;u,v)$. Typically, we would write

$$F_k(\xi) = \sum_{\rho\in\mathcal{L}} F_k^{\iota(e,\rho)} \phi_\rho(\xi)$$

so that

$$J_F(\xi)_{k,m} = \frac{\partial \xi_m}{\partial F}_k(\xi) = \sum_{\rho\in\mathcal{L}} F_k^{\iota(e,\rho)} \frac{\partial \xi_m}{\partial \phi}_\rho(\xi).$$

The above expression for a_e is then a trilinear expression in v_k, w_k and F_k. The cost of evaluating this in terms of element matrices would thus be on the order of $|\mathcal{L}|^3$.

The other way to evaluate multilinear forms can be more efficient than using element matrices. The cost of evaluating $J_F(\xi)$ as above is only on the order of $|\Xi|\cdot|\mathcal{L}|$. The amount

of work involved in the inversion and transpose of J is independent of $|\Xi|$ and $|\mathcal{L}|$. Therefore, the work estimate for the latter approach remains of the order $|\Xi| \cdot |\mathcal{L}|$.

We can apply this approach to the trilinear form $c(\cdot,\cdot,\cdot)$ as well:

$$\begin{aligned} c_e(\mathbf{u},\mathbf{v},\mathbf{w}) = & \sum_{\xi\in\Xi} \omega_\xi \sum_{j,k=1}^{d} \left(\sum_{\lambda\in\mathcal{L}} u_j^{\iota(e,\lambda)} \phi_\lambda(\xi) \right) \times \\ & \left(J_F^{-1}(\xi)^t \sum_{\lambda\in\mathcal{L}} v_k^{\iota(e,\lambda)} \nabla\phi_\lambda(\xi) \right)_j \left(\sum_{\lambda\in\mathcal{L}} w_k^{\iota(e,\lambda)} \phi_\lambda(\xi) \right) \det(J_F(\xi)) \ . \end{aligned} \tag{59}$$

This has the same order $|\Xi| \cdot |\mathcal{L}|$ work estimate. Thus the asymptotic work estimate does not change with the introduction of isoparametric elements.

THEOREM The local κ-linear forms for isoparametric elements can be computed in an amount of work proportional to the product of κ, the number of quadrature points, and the number of degrees of freedom of the local basis functions.

5 A compiler for variational forms

Suppose that we want to develop a compiler for a language of forms as described in Section 1. The first thing to realize is that the mapping back to the reference element induces a change for the derivatives. That is `D` gets translated to $G(D)M(D)D$ with the appropriate multiplication operator $M(D)$, where $G(D)$ represents a matrix involving the transformation in some way. This will of course involve the Jacobian, but the transformation can be more complicated for some finite elements. The following example

$$(\mathtt{u}\ \mathtt{m}_1(\mathtt{D}_1\mathtt{v})\ \mathtt{m}_2(\mathtt{D}_2\mathtt{w})) \tag{60}$$

gets converted to

$$(u\ m_1(G(D_1)M(D_1)(D_1 v))\ m_2(G(D_2)M(D_2)(D_2 w)) \tag{61}$$

as an intermediate representation. Let us presume that expressions such as Dw can be evaluated at quadrature points efficiently by some other process, so that we only have to generate code to resolve the various multiplications and G's, and collect like terms. However, we need to reduce vector operations to sums (or loops) of scalar operations.

We could of course just follow the expressions in (61) by rote and generate the corresponding code as loops. But we have seen that this will be far from optimal. Thus we might recognize an inner-product that would allow us to transform (61) to

$$(u\ m_1 G_2 M_3((D_1 v)\ m_2(D_2 w))) \tag{62}$$

where G_2 is some new expression involving the Jacobian.

To begin with, let us determine a general form for a differential operator of the type we will try to compile. Let us consider general expressions where D will map tensors u of arity $\mathbf{ar}(u)$. That is, u has individual components u_σ where σ is a multi-index $\sigma = (\sigma_1, \ldots, \mathbf{ar}(u))$.

Table 1: Differential operators of low arity. In the table, $d = \mathbf{dd}(u)$, and any value of $d \geq 1$ is valid; I_d denotes the $d \times d$ identity matrix.

$\mathbf{ar}(D)$: $\mathbf{dr}(D)$	$\mathbf{ar}(u)$: $\mathbf{dr}(u)$	form of C	form of (62)	examples: $Du; C$
0: 0	0 : 0	d-vector	$C \cdot \nabla u$	(1) $u_{,1}; C = (1, 0, \ldots, 0)$ (2) $u_{,d}; C = (0, \ldots, 0, 1)$
0: 0	1: $d' \geq 1$	$d' \times d$ matrix	$C : \nabla \mathbf{u}$	(3) $\nabla \cdot \mathbf{u}; C = I_d$ $(d' = d)$ (4) curl $\mathbf{u} = u_{2,1} - u_{1,2}$; $C = \begin{pmatrix} 0 & 1 \\ -1 & 0 \end{pmatrix}$ $(d = d' = 2)$
1: $d' \geq 1$	0: 0	$d' \times d$ matrix	$C * \nabla u$	(5) $\nabla u; C = I_d$ $(d' = d)$ (6) **curl** $u = (u_{,2}, -u_{,1})$; $C = \begin{pmatrix} 0 & 1 \\ -1 & 0 \end{pmatrix}$ $(d = d' = 2)$ (7) $\nabla \times u$ $(d = d' = 3)$; $C = \begin{pmatrix} 0 & -1 & 1 \\ 1 & 0 & -1 \\ -1 & 1 & 0 \end{pmatrix}$
1: $d'' \geq 1$	1: $d' \geq 1$	$C_\mu = d' \times d$ matrix	$[C_\mu : \nabla \mathbf{u}]$	(8) **div** $\mathbf{u}; C_\mu = I_d$ $(d' = d)$
2: $(d_3, d_2) \geq (1, 1)$	1: $d_1 \geq 1$	$C_\mu = d_1 \times d$ matrix	$[C_\mu : \nabla \mathbf{u}]$	(9) $\nabla \mathbf{u}; C = (\delta_{\mu_1,\sigma} \delta_{\mu_2,\ell})$ (10) $\nabla \mathbf{u}^t; C = (\delta_{\mu_2,\sigma} \delta_{\mu_1,\ell})$ (11) $2\sigma(\mathbf{u}) = \nabla \mathbf{u} + \nabla \mathbf{u}^t$; $C = (\delta_{\mu_2,\sigma} \delta_{\mu_1,\ell}) + (\delta_{\mu_1,\sigma} \delta_{\mu_2,\ell})$

The range of the indices σ_i satisfy $1 \leq \sigma_i \leq \mathbf{dr}(u)_i$ (or equivalently $\sigma \leq \mathbf{dr}(u)$) where $\mathbf{dr}(u)$ denotes a multi-index of the same arity as u and give the number of dimensions for each component. For example, vector functions have arity one ($\mathbf{ar}(u) = 1$) and $\mathbf{dr}(u) = \mathbf{dr}(u)_1$ is the dimension of the vector space that is the image of u. If u denotes a matrix, then $\mathbf{ar}(u) = 2$. A scalar valued function has $\mathbf{ar}(u) = 0$.

With this definition, each differential operator maps functions of one arity to another. Thus we can define $\mathbf{ar}(D)$ to be the arity of the functions of the form Du, i.e., $\mathbf{ar}(D) = \mathbf{ar}(Du)$. Similarly, $\mathbf{ad}(D)$ is the arity of the functions u for which the the differential operator is defined, that is, $\mathbf{ad}(D) = \mathbf{ar}(u)$ where Du is defined. Finally, all this also depends on the domain of u. That is, the differentiation can only be in the variables for which u is defined. Thus, for example, the two-dimensional curl operator $u_{2,1} - u_{1,2}$ depends on $\mathbf{dd}(u)$ as well as $\mathbf{ar}(u)$, and $\mathbf{ar}(\text{curl}) = 0$.

With these definitions in hand, we now easily express a general first-order, linear differential operator as

$$(Du)_\mu = \sum_{\sigma,\ell} C_{\mu\sigma\ell} u_{\sigma,\ell} \tag{63}$$

where the ranges are $1 \leq \ell \leq \mathbf{dd}(u)$, $0 \leq \sigma \leq \mathbf{dr}(u)$, and $0 \leq \mu \leq \mathbf{dr}(D)$. This general tensor expression can be quite high dimensional. However, in low dimensional cases, the tensor multiplication can be given a simpler interpretation, as collected in Table 1.

Some things emerge from the examples in Table 1. First of all, since there are eleven different operators, there could be a large number of combinations possible. So it would be useful to automate the generation of forms. However, the general formula (63) would not

be very efficient for a general solution for these examples. We see that the matrices C are quite large yet very sparse and structured. Thus, it seems reasonable to build in some rules for dealing with each of these terms, together with possible interactions.

6 Conclusions

There are several general conclusions that we can draw.

- It is useful to have a language for spaces of finite element functions, and to have this tied to a geometry language. The key feature integrating these two languages is a projection operator defined using geometric concepts like "interior" and "boundary" for domains. The projection operators apply to functions and operators in a consistent way.
- There are several ways that a given computation can be done, and there is no universal algorithm that is optimal. It may be useful to provide options that can be optimized for particular architectures.
- Symmetry may play a role; the Jacobian is always symmetric, and matrices derived from it may be symmetric. Symmetric bilinear forms generate symmetric matrices.
- Some vector-valued problems generate matrices with special properties, such as being block diagonal. This can lead to an improvement in performance of an order of magnitude if recognized and exploited.
- The mapping of nodal values to values (and gradients) at quadrature points is essential. If this can be done efficiently, then the quadrature approach is efficient for isoparametrics.

References

[1] Alnæs, M. S., and Logg, A. UFL Specification and User Manual 0.1. http://www.fenics.org/pub/documents/ufl/ufl-user-manual/ufl-user-manual.pdf, 2009.

[2] Bagheri, B., and Scott, L. R. About Analysa. Research Report UC/CS TR-2004-09, Dept. Comp. Sci., Univ. Chicago, 2004.

[3] Kirby, R. C., Knepley, M., and Scott, L. R. Evaluation of the action of finite element operators. Research Report UC/CS TR-2010-8, Dept. Comp. Sci., Univ. Chicago, 2010.