Basis function representation of fields on meshes

July 6, 2010

1 Introduction

This document aims to describe a data model for storing discrete functions on meshes. It deliberately focusses on what data needs to be stored and does not attempt to describe, for example, what the netcdf fields would need to be.

The premise of this document is that an unstructured mesh data format has two functions: representation and facilitation of interoperability. The essence of the representation function is that the representation of data in the application writing the file should be translated losslessly into the data format. That is to say, the variable placement and values of all the fields output need not change for them to be represented in the file[1].

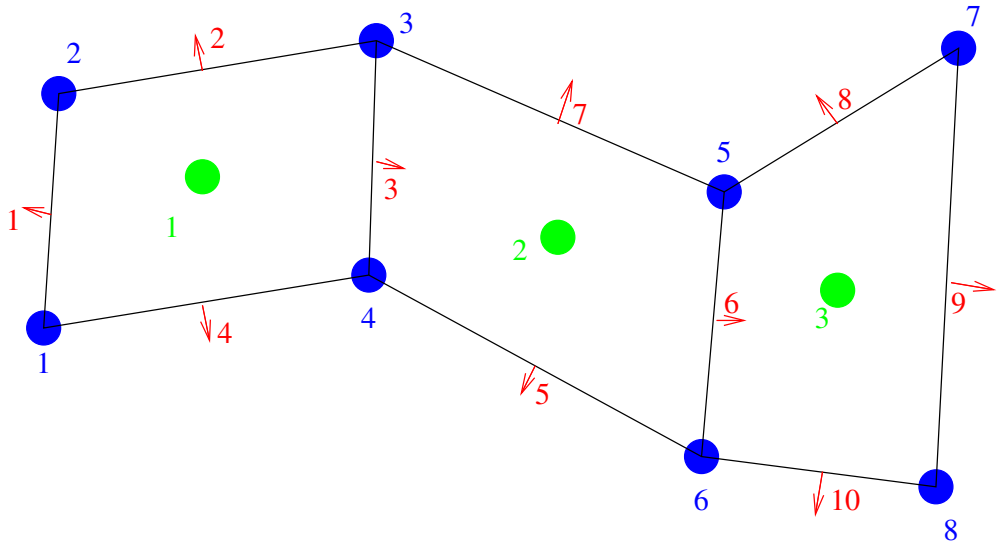
The facilitation of interoperability function is in many respects the reverse of this: tools may only be able to operate on data in a particular form. For example many visualisation tools are only capable of acting on linear or bilinear collocated data. The data format should therefore carry with it the information required to transform the data into a different representation (and this transformation may be lossy).

Here we first examine how fields can be represented in terms of the basis functions for their underlying elements. It is argued that this data model allows the representation function to be fulfilled for a very large range of function space choices. The examples given are all for quadrilateral elements, in response to the query in the webex about how to represent C-grid. The data model obviously extends to other element shapes.

The final part of the document examines how this representation facilitates interoperability between data in different function spaces and therefore between tools with different function space expectations.

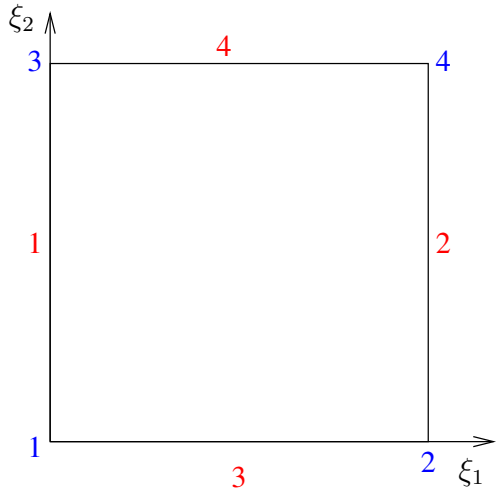
[1] Of course an application may choose to output in a lossy way for its own purposes but this should not be forced by the file format

2 Example mesh



3 The reference quadrilateral

Every aspect of a quadrilateral element is defined with respect to a reference quadrilateral. This is the unit quadrilateral in two local coordinates ξ_1 and ξ_2[2]. The reference quadrilateral has a local numbering of vertices and edges with respect to which all actual values in the mesh are recorded. Without making a particular statement about what the reference quadrilateral should be, the following is the one which will be used in this document:



[2]some authors prefer the reference coordinates to run from -1 to 1 rather than 0 to 1

The vertices of the reference element can be identified with their local coordinates:

Node	(ξ_1, ξ_2)
1	(0, 0)
2	(1, 0)
3	(0, 1)
4	(1, 1)

4 Defining the topology

The topology of the mesh is determined by listing which vertices make up each element this is referred to as the element node list. These are listed in local node order. Note that the *representation* of the topology is not unique because the numbering could start with any node on each element. It is possible to derive the edges from this topology. Doing the reverse (listing the edges and deriving the elements) is harder.

If the elements are numbered in the order given by the green numbers, then the topology could be given by the 2D array:

Element	N_1	N_2	N_3	N_4
1	1	4	2	3
2	3	4	5	7
3	8	7	6	5

5 Defining the elements

The first basis functions which we define will be those used to define bilinear elements. This is the element used in the "A"-grid or, in finite element terminology, the Q1 element.

The four basis functions, corresponding to the four nodes are:

$$\phi_1 = (1-\xi_1)(1-\xi_2)$$
$$\phi_2 = \xi_1(1-\xi_2)$$
$$\phi_3 = (1-\xi_1)\xi_2$$
$$\phi_4 = \xi_1\xi_2$$

It is trivial, but important, to observe that each of these basis functions evaluates to 1 at the corresponding local node and 0 elsewhere.

6 Coordinates

The topology does not of itself require coordinates. It merely describes adjacency relationships of various sorts. The coordinate field is naturally actually defined everywhere (every point in each element has coordinates) and the coordinate field may not be a linear function of the local coordinates (this accounts for "bendy" elements). In the simple case of bilinear coordinates on quadrilateral elements, the coordinate field is given by the Q1 element described above. This requires us to associate a set of coordinates with each node in the mesh. For example.

Node	(x_1, x_2)
1	(2.1, 3.0)
2	(2.3, 7.0)
3	(6.5, 7.5)
4	(6.4, 3.3)
5	(11.2, 6.0)
6	(11.0, 2.0)
7	(14.2, 8.0)
8	(14.0, 1.8)

This information is sufficient to determine the coordinates of any object in the mesh. In particular it is sufficient to determine the coordinates of the nodes of any element type. To see why this is true, we can look at the example of the Q0 element in the next section.

7 C-grid scalars: the Q0 element

On the "C"-grid, scalars are piecewise constant over each element. This can be represented by the Q0 element which has a single node located at the barycentre of the element. Its local coordinates are therefore given by:

Node	(ξ_1, ξ_2)
1	(0.5, 0.5)

And the corresponding basis function on each element is:

$$\psi_1 = 1.0$$

The element node list associated with this field type is also very trivial as there is a one-to-one mapping between nodes and elements:

Element	N_1
1	1
2	2
3	3

Now let's substantiate the claim that it is not necessary to store the coordinates of the scalar nodes. Suppose we wish to know the position of global scalar node 1. First we look up element 1 in the Q0 element node list and determine that this is also local node 1.

Now local node 1 has coordinates $(0.5, 0.5)$. We now need to evaluate the coordinate field at those local coordinates. By looking up the Q1 element node list we determine that the nodes in element 1 are 1, 4, 2, 3. To evaluate the coordinate at $(0.5, 0.5)$ we therefore have:

$$\begin{aligned} x &= \sum_{i=1}^{4} x_i \phi_i(0.5, 0.5) \\ &= x_1 \times 0.25 + x_4 \times 0.25 + x_2 \times 0.25 + x_3 \times 0.25 \\ &= (2.1, 3.0) \times 0.25 + (6.4, 3.3) \times 0.25 + (2.3, 7.0) \times 0.25 + (6.5, 7.5) \times 0.25 \\ &= (4.325, 5.2) \end{aligned}$$

As one would expect, the value of the coordinate at the element centre is the mean of the corner values, however writing it in this way produces a mechanism which will work to determine the coordinates of the nodes of any discretisation type.

8 C-grid vectors

The C-grid velocity basis functions are slightly trickier as they are vector-valued. However the same principles apply. There are four nodes on each quadrilateral, each located in the middle of one of the edges. Using the reference quadrilateral above, we can write the local coordinates:

Node	(ξ_1, ξ_2)
1	(0, 0.5)
2	(1, 0.5)
3	(0.5, 0)
4	(0.5, 1)

For the basis functions, we need to know which way the global normal vector, $\vec{N}$ points on each face. This is point 1.f.i. in the Unstructured Grid Data Set Standard Requirements. In the diagram in section 2 we have assumed that the global normal on an interior edge points towards the element with higher number and that the global normal points outwards on all boundaries. Other conventions are possible and do not change the data

model. We also need to define the local normal $\vec{n}$, which points outward on each element edge. In turn this allows us to define:

$$s = \vec{N} \cdot \vec{n}.$$

On each face, s is a switch which returns ± 1 according to whether the local and global normals on that face point in the same or opposite directions. The next issue we need to consider is that the change of coordinates from local (ξ_i) to physical ($\vec{x}$) applies to velocity as well as to position. If we consider the definition of velocity:

$$u_i = \frac{\mathrm{d}x_i}{\mathrm{d}t}$$

then we can apply the chain rule to acquire the velocity in terms of the local coordinates:

$$u_i = \frac{\mathrm{d}x_i}{\mathrm{d}\xi_j}\frac{\mathrm{d}\xi_j}{\mathrm{d}t}$$

The matrix:

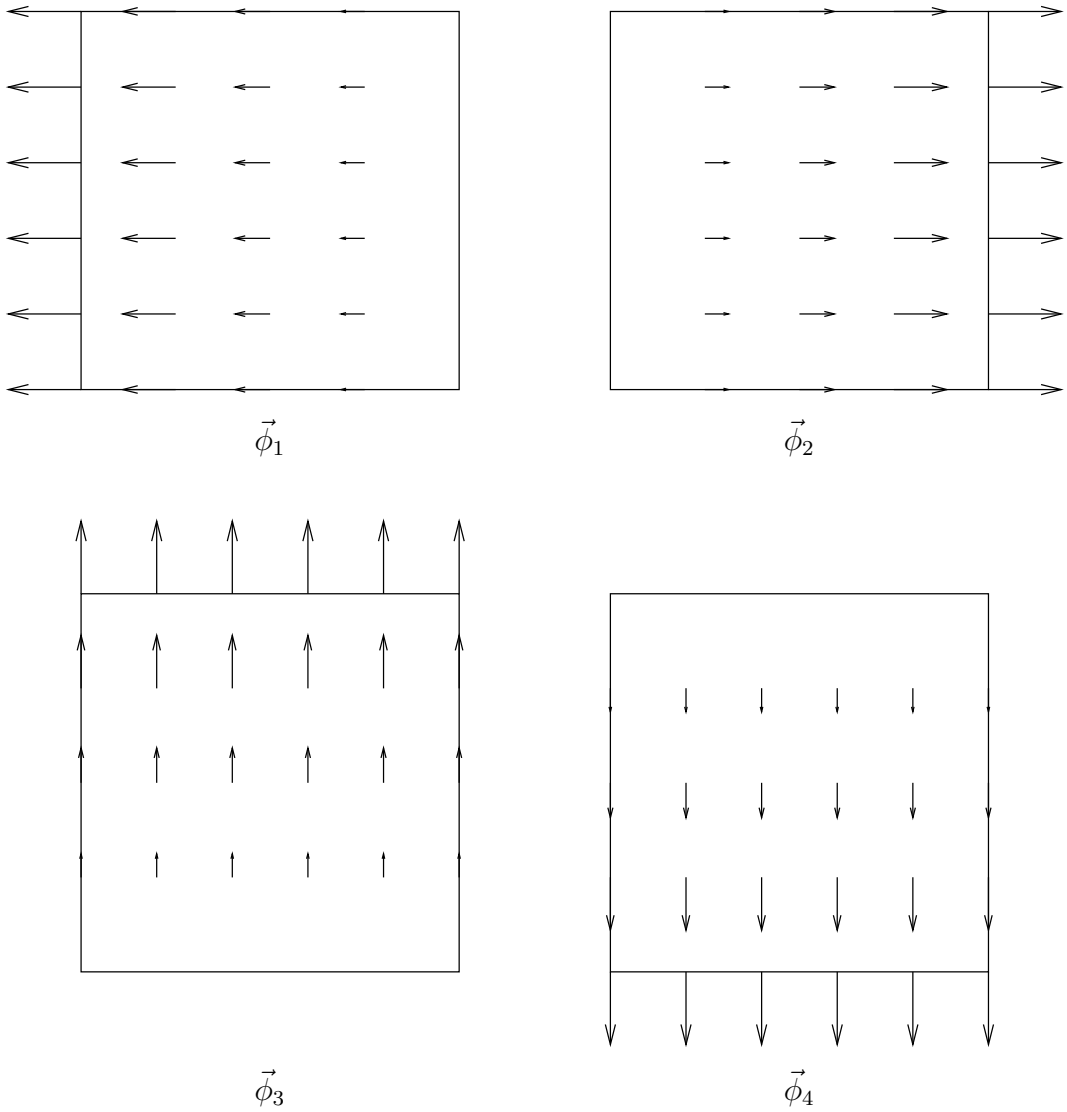
$$J = \frac{\mathrm{d}x_i}{\mathrm{d}\xi_j}$$

is known as the element Jacobian and can be calculated using the coordinate field on the element.

This now allows us to define the basis functions:

$$\begin{aligned}
\vec{\phi}_1 &= sJ \cdot [(\xi_1 - 1), 0] \\
\vec{\phi}_2 &= sJ \cdot [\xi_1, 0] \\
\vec{\phi}_3 &= sJ \cdot [0, (\xi_2 - 1)] \\
\vec{\phi}_4 &= sJ \cdot [0, \xi_2]
\end{aligned}$$

The basis functions evaluated on the unit square on the assumption that the global normals align with the local normals (ie $s = 1$ on all edges) look like:



Finally we can write the element node list for the C-grid velocity on our example mesh:

Element	N_1	N_2	N_3	N_4
1	1	3	2	4
2	7	5	3	6
3	10	8	9	6

It will be observed that this basis function representation is quite different from the way in which a C-grid finite difference or finite volume model is usually implemented. It should be remembered that this data model only attempts to convey information which could be used by other tools to interpret the model output. It has no impact on how the model itself might be written.

9 Components of the data model

9.1 Elements

The data model illustrated above can be conceptualised as follows. The smallest object is the element. This is the description of the nodes and basis functions for a particular variable placement on the reference element. An element consists of (at least):

- A list of the local coordinates of the nodes.
- A list of the basis functions.

It is likely that it will also be useful to record some more information about the element:

- dimension.
- geometric shape, number of vertices, faces, edges etc.
- continuity between elements: continuous or discontinuous. Some elements, such as C-grid, are partially continuous and this might be useful to record as well.

There are two compatible ways that elements could be specified. First, a list of common elements could be specified in the standard. This would enable many applications to simply specify which element they are using without giving the full element specification. So, for example, a model could specify quadrilateral c-grid. Second, the application could write the full element specification into the data file. This option allows for built-in extensibility as the data format can be used to specify any element for which basis functions can be written.

9.2 Function spaces

Given an element and an element node list, it is possible to map between global degrees of freedom and local elements. This is sufficient to define the space of functions for any field value. To accommodate meshes with mixtures of element shapes or element degrees, it should be possible to list a number of elements and for each element to provide an element node list for the parts of the mesh to which it applies.

In this way, the Element specifies a local basis for functions on a single element and the function space glues together elements to provide a global basis for functions over the whole mesh.

There should be able to be any number of function spaces as different fields may be stored on different spaces (for example in the C-Grid, the velocity, scalars and coordinate fields are each in different function spaces).

Once again it should be noted that from the topology it is possible to generate a list of edges, node adjacency information and element adjacency information. In this way the topology of the mesh and the node element list of each function space is a minimal set of information which describes all of the data relationships on the mesh.

9.3 Fields

A field associates a list of values with the basis functions in a function space. For example for the c-grid velocity it is the list of edge centre normal velocity components. However in the data model it's "just" a list of coefficients for the basis functions of the function space. In this way the description is totally general to all element choices.

10 Boundary conditions

The specification of boundary conditions is made easy by noticing that function spaces can be defined on the facets of the elements by using elements with dimension 1 less than the main element. In our example above, the domain boundary could be defined as a set of 1-dimensional elements. The reference 1D element is a unit interval with a single local coordinate which runs from 0 to 1. The boundaries of the Q1 element become the linear P1 element with two nodes:

Node	(ξ_1)
1	(0)
2	(1)

and basis functions:

$$\phi_1 = 1 - \xi_1$$
$$\phi_2 = \xi_1$$

A node element map describing the boundary might be:

Element	N_1	N_2
1	1	2
2	2	3
4	1	4
5	4	6
7	3	5
8	5	7
9	7	8
10	6	8

If contiguous numbering of the boundary is desired then either the boundary edges could be constrained to be the lowest numbered or a sepa-

rate boundary numbering could be adopted and mapped back to the edge numbers.

11 Interoperability

The interoperability benefits of the basis function representation stem from the fact that the value of each field is known everywhere. In the simplest case, this means that the value of any field can be evaluated at any point so model output can be compared pointwise. Another very common requirement is for visualisation. Suppose we have a visualisation tool which is capable only of representing fields in Q1 (ie bilinear nodal values) but our model has produced Q0 output (ie cell-constant). We can produce a Q1 field for visualisation by Galerkin projection by solving the equation:

$$\int_\Omega \phi_i \phi_j f_j \mathrm{d}x = \int_\Omega \phi_i \psi_k g_k \mathrm{d}x$$

where f is the Q1 field with basis functions ϕ and g is the Q0 field with basis functions ψ. Summation is carried out over repeated indices. Note that because the field carries with it information about the basis functions, we can construct generic tools which perform this projection between any two spaces (indeed the ICOM group has some of these tools already although we are hampered by not having a good file format for a wide range of these to apply).

The pure Galerkin projection above preserves the integral of the field over element patches (even more locally for suitable choices of basis function) and minimises the L2 error between f and g. More advanced projections are also possible to preserve properties such as boundedness and/or divergence-freeness. Using a supermesh it is even possible to use the basis functions to project between fields on topologically distinct meshes. This is important if we wish to answer questions like "what is the difference between the output of these two models" in a rigorous way. Bounded projection and supermeshes are covered in Farrell PE, Piggott MD, Pain CC, Gorman GJ, Wilson CR, Conservative interpolation between unstructured meshes via supermesh construction, Comp. Meth. Appl. Mech. Eng., 198, 2632-2642, 2009. doi:10.1016/j.cma.2009.03.004 .