Mathematical Representation of FEM Module Operations

Tyler Olsen

July 28, 2016

Contents

1 Introduction **1**

2 Element Class **2**
2.1 Basic Information . . . 2
2.2 Data Arrays . . . 2

3 Parent Elements **3**
3.1 Linear Triangle (TRI3) . . . 3
3.1.1 ElementTraits for TRI3 . . . 3
3.1.2 Parent nodes and shape functions . . . 3
3.1.3 Quadrature points and weights . . . 4
3.2 Linear Tetrahedron (TET4) . . . 4
3.2.1 ElementTraits for TET4 . . . 4
3.2.2 Parent nodes and shape functions . . . 4
3.2.3 Quadrature points and weights . . . 4
3.3 Linear Quadrilateral (QUAD4) . . . 4
3.3.1 ElementTraits for QUAD4 . . . 5
3.3.2 Parent nodes and shape functions . . . 5
3.3.3 Quadrature points and weights . . . 5
3.4 Linear Hexahedron (HEX8) . . . 5
3.4.1 ElementTraits for HEX8 . . . 5
3.4.2 Parent nodes and shape functions . . . 6
3.4.3 Quadrature points and weights . . . 6

4 QPData structure **6**
4.1 Isoparametric mapping recap . . . 6
4.2 Computing the Jacobian . . . 7
4.3 Computing spatial derivatives . . . 7
4.4 Accessing QPData Members . . . 7

1 Introduction

This document builds on the FEM introduction given in the Yet Another Finite Element Library (YAFEL) slide deck that I presented to Anthony at the beginning of July. Therein, a prototypical

problem (steady-state heat transfer) is mathematically developed, and key aspects of its implementation via the YAFEL library are demonstrated. However, the presentation does not fully present the details of what YAFEL is doing behind the scenes to allow such a compact and convenient representation of the problem. In this document, I aim to open the hood and show the mathematical representation and point to where and how this is implemented in the X2 framework. My aim for this document is that you will be able to understand what problems each data structure in X2 address.

2 Element Class

The element class holds data that is common to every every element of a given type. One of these objects is instantiated within the ElementFactory for each supported element type. The following sections describe the information provided by the Element class.

2.1 Basic Information

This information is taken from the ElementTraits class to allow access at runtime. These quantities are available directly as members of the Element class.

- n_nodes : Number of nodes in an element
- n_dims : The topological dimension of the element
- n_quadpoints : The number of quadrature points used for numerical integration
- name : A string representation of the element type

2.2 Data Arrays

The following data arrays are available using the provided accessor functions. The storage for these arrays is statically allocated and managed by the ElementFactory object. Users should not attempt to create their own Element objects; rather, they should always use the one provided by the `ElementFactory::get_element` function. If, eventually, enough different types of elements are supported that it makes sense to heap-allocate these buffers, this should also be done by the ElementFactory object.

Array name	Math name	Data size	accessor function
Shape values	$N^A(\xi\|_{qp})$	$n_quadpoints \times n_nodes$	`shape_value(...)`
Shape gradients	$\frac{\partial N^A}{\partial \xi_j}\Big\|_{qp}$	$n_quadpoints \times n_nodes \times n_dims$	`shape_grad_xi(...)`
Quadrature point	$\xi\|_{qp}$	$n_quadpoints \times n_dims$	`quadrature_point(...)`
Quadrature weight	w_{qp}	$n_quadpoints$	`quadrature_weight(...)`

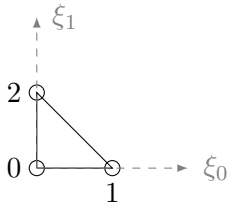
3 Parent Elements

This section presents each of the supported elements in their parent forms, defines the assumed node ordering, gives the shape functions, and the default (currently only) quadrature rule. The two-dimensional elements have accompanying images to clarify their definitions. The three-dimensional analogs have a description that hopefully is sufficient to define the element. If in doubt, these node orderings follow the ordering conventions of the GMSH program and can be found in their documentation. If you want to implement new element types, this is the information that must be supplied in order for the X2 FEM machinery to take over. Note that you do NOT need to implement the gradients (with respect to ξ) of the shape functions. This is taken care of automatically using Dual Number automatic differentiation in the "make_element" function located in "element_impl.hpp".

A note about notation: superscripts indicate that the quantity refers to a node in the parent element. For example, ξ^0 is the location in ξ-space of node 0. Subscripts indicate a component of a vector/tensor quantity. For example, ξ_0 is the first component of a vector coordinate ξ. Combining the two has predictable results: ξ_0^3 is the first component of the location of node 3. Lastly, as you may have noticed, I'm following the convention of zero-based indexing. I find that although it is unnatural to read the math in this way, it is far less error-prone to translate into C-family code with zero-based array indexing.

3.1 Linear Triangle (TRI3)

This is a 45-degree right triangle in the first quadrant of two-dimensional ξ-space.



3.1.1 ElementTraits for TRI3

Trait Name	Trait Value
n_dims	2
n_nodes	3
n_quadpoints	1
name	"TRI3"

3.1.2 Parent nodes and shape functions

$$\xi^0 = (0,0) \qquad N^0(\xi) = 1 - \xi_0 - \xi_1$$
$$\xi^1 = (1,0) \qquad N^1(\xi) = \xi_0$$
$$\xi^2 = (0,1) \qquad N^2(\xi) = \xi_1$$

3.1.3 Quadrature points and weights

Points	Weights
$(1/3, 1/3)$	$1/2$

3.2 Linear Tetrahedron (TET4)

This parent element is a 45-degree tetrahedron in the first octant of three-dimensional ξ-space. The node ordering is analogous to the TRI3, with an additional node on the ξ_2 axis at $\xi^3 = (0, 0, 1)$.

3.2.1 ElementTraits for TET4

Trait Name	Trait Value
n_dims	3
n_nodes	4
n_quadpoints	1
name	"TET4"

3.2.2 Parent nodes and shape functions

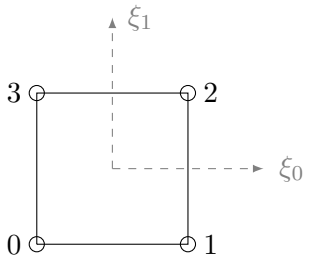
$$\begin{aligned} \xi^0 &= (0, 0, 0) & N^0(\xi) &= 1 - \xi_0 - \xi_1 - \xi_2 \\ \xi^1 &= (1, 0, 0) & N^1(\xi) &= \xi_0 \\ \xi^2 &= (0, 1, 0) & N^2(\xi) &= \xi_1 \\ \xi^3 &= (0, 0, 1) & N^3(\xi) &= \xi_2 \end{aligned}$$

3.2.3 Quadrature points and weights

Points	Weights
$(1/4, 1/4)$	$1/4$

3.3 Linear Quadrilateral (QUAD4)

This parent element is a bi-unit square in two-dimensional ξ-space.



3.3.1 ElementTraits for QUAD4

Trait Name	Trait Value
n_dims	2
n_nodes	4
n_quadpoints	4
name	"QUAD4"

3.3.2 Parent nodes and shape functions

$$\xi^0 = (-1, -1) \qquad N^0(\xi) = \frac{1}{4}(1 - \xi_0)(1 - \xi_1)$$

$$\xi^1 = (+1, -1) \qquad N^1(\xi) = \frac{1}{4}(1 + \xi_0)(1 - \xi_1)$$

$$\xi^2 = (+1, +1) \qquad N^2(\xi) = \frac{1}{4}(1 + \xi_0)(1 + \xi_1)$$

$$\xi^3 = (-1, +1) \qquad N^3(\xi) = \frac{1}{4}(1 - \xi_0)(1 + \xi_1)$$

3.3.3 Quadrature points and weights

	Points	Weights
	$(-a, -a)$	1
Let $a = 1/\sqrt{3}$:	$(a, -a)$	1
	(a, a)	1
	$(-a, a)$	1

3.4 Linear Hexahedron (HEX8)

This parent element is a bi-unit cube in three-dimensional ξ-space. The ordering is the same as the QUAD4 at two different planes, $\xi_2 = \pm 1$; i.e. nodes 0-3 are a QUAD4 at $\xi_2 = -1$ and nodes 4-7 are a QUAD4 at $\xi_2 = 1$.

3.4.1 ElementTraits for HEX8

Trait Name	Trait Value
n_dims	3
n_nodes	8
n_quadpoints	8
name	"HEX8"

3.4.2 Parent nodes and shape functions

$$
\begin{aligned}
\xi^0 &= (-1,-1,-1) & N^0(\xi) &= \frac{1}{8}(1-\xi_0)(1-\xi_1)(1-\xi_2)\\
\xi^1 &= (+1,-1,-1) & N^1(\xi) &= \frac{1}{8}(1+\xi_0)(1-\xi_1)(1-\xi_2)\\
\xi^2 &= (+1,+1,-1) & N^2(\xi) &= \frac{1}{8}(1+\xi_0)(1+\xi_1)(1-\xi_2)\\
\xi^3 &= (-1,+1,-1) & N^3(\xi) &= \frac{1}{8}(1-\xi_0)(1+\xi_1)(1-\xi_2)\\
\xi^4 &= (-1,-1,+1) & N^4(\xi) &= \frac{1}{8}(1-\xi_0)(1-\xi_1)(1+\xi_2)\\
\xi^5 &= (+1,-1,+1) & N^5(\xi) &= \frac{1}{8}(1+\xi_0)(1-\xi_1)(1+\xi_2)\\
\xi^6 &= (+1,+1,+1) & N^6(\xi) &= \frac{1}{8}(1+\xi_0)(1+\xi_1)(1+\xi_2)\\
\xi^7 &= (-1,+1,+1) & N^7(\xi) &= \frac{1}{8}(1-\xi_0)(1+\xi_1)(1+\xi_2)
\end{aligned}
$$

3.4.3 Quadrature points and weights

Let $a = 1/\sqrt{3}$:

Points	Weights
$(-a,-a,-a)$	1
$(a,-a,-a)$	1
$(a,a,-a)$	1
$(-a,a,-a)$	1
$(-a,-a,a)$	1
$(a,-a,a)$	1
(a,a,a)	1
$(-a,a,a)$	1

4 QPData structure

This structure is responsible for providing element-specific geometric quantities that require node positions in the physical mesh. The two quantities of primary importance are the jacobian transformation of the isoparametric mapping and the shape function gradients with respect to spatial coordinates ($\mathbf{x}$-space).

4.1 Isoparametric mapping recap

Recall from the YAFEL presentation that space is interpolated using shape functions. Namely, there is a one-to-one correspondence of each ξ location in the parent element to an $\mathbf{x}$ location in

the physical one. This is done with the mapping function ϕ.

$$\mathbf{x} = \phi(\xi) = \sum_{A \in [0, n_nodes)} N^A(\xi) * \mathbf{x}^A \tag{1}$$

4.2 Computing the Jacobian

The jacobian of the mapping can be computed by differentiating each component of the mapping function (1) with respect to each component of ξ.

$$J_{ij} = \frac{\partial \phi_i}{\partial \xi_j} = \sum_{A \in [0, n_nodes)} \frac{\partial N^A(\xi)}{\partial \xi_j} * x_i^A \tag{2}$$

These components are part of a second-order tensor defined by

$$\mathbf{J} = J_{ij} \mathbf{e}_i \otimes \mathbf{e}_j \tag{3}$$

assuming Einstein's summation convention with cartesian basis vectors $\mathbf{e}_i$.

This equation is implemented as part of the function "qpdata_udpate" in "QPData.cpp".

4.3 Computing spatial derivatives

The spatial derivatives can be defined using the chain rule.

$$\frac{\partial N^A}{\partial x_j} = \sum_k \frac{\partial N^A}{\partial \xi_k} \frac{\partial \xi_k}{\partial x_j} \tag{4}$$

The final term in this expression can be recognized as components of the jacobian of the inverse mapping function, $\frac{\partial \phi_i^{-1}}{\partial x_j}$. Generally speaking, this is impossible to compute analytically, but this is exactly the same as the tensor inverse of the "forward" jacobian under the assumption of an invertible mapping function (we'd be stuck either way without this assumption).

Thus, we simply invert $\mathbf{J}$ and apply the following relationship to compute shape function spatial derivatives.

$$\frac{\partial N^A}{\partial x_j} = \sum_k \frac{\partial N^A}{\partial \xi_k} * (\mathbf{J}^{-1})_{kj} \tag{5}$$

This equation is implemented as part of the function "qpdata_udpate" in "QPData.cpp".

4.4 Accessing QPData Members

This class has similar functionality to the Element class. That is, it has some directly-accessible basic information and some data arrays accessible via accessor functions. An important distinction between the two, however, is that the QPData structure only holds information at a single quadrature point. Thus, the "`qpdata_update`" function in "QPData.cpp" must be called at each

quadrature point. Failing to do so will result in incorrect element calculations. This function requires a pointer to a QPData object, an Element object, and a vector of element node coordinates. The following table shows what information is avaialble and how to extract it from QPData.

Data name	Math name	Data Size	Access method
Num nodes	n_nodes	1	QP.n_nodes
Num dims	n_dims	1	QP.n_dims
Jacobian determinant	$det(\mathbf{J})$	1	QP.detJ
Jacobian	$\mathbf{J}$	$n_dims \times n_dims$	`qp_jacobian(...)`
Shape gradient	$\frac{\partial N^A}{\partial x_j}$	$n_nodes \times n_dims$	`qp_shape_grad(...)`

As with the Element object, the data arrays are stored in statically-allocated buffers owned by the QPData object. The buffers (being statically allocated) are sized to accommodate the requirements of the largest supported element. The buffer sizing is handled automatically at compile-time by using the same set of metafunctions as the ElementFactory object to examine the list of supported elements.

Design Note: Due to their similarity, it seems like the Element and QPData classes should be merged. However, this would prove to be a mistake if you consider parallelization. Each worker thread would be working on separate elements, so each worker needs its own QPData to hold the element-specific information. After construction, the data in Element is read-only, so it can be safely shared across threads.