White Paper:
Piecewise Linear Shape Functions on Polygon type cells

June, 2019

Jan Vermaak

Rev 1.00

1 Introduction - Piecewise linear shape functions on a 2D triangle

For a two dimensional simulation using triangular elements we seek to map a triangle in cartesian space to a reference triangle in natural coordinates. We do this because we can develop a method to perform integration or differentation for the reference triangle that can be mapped to a triangle of any shape and location. An example of the two triangles in different coordinate space is shown in Figure 1.

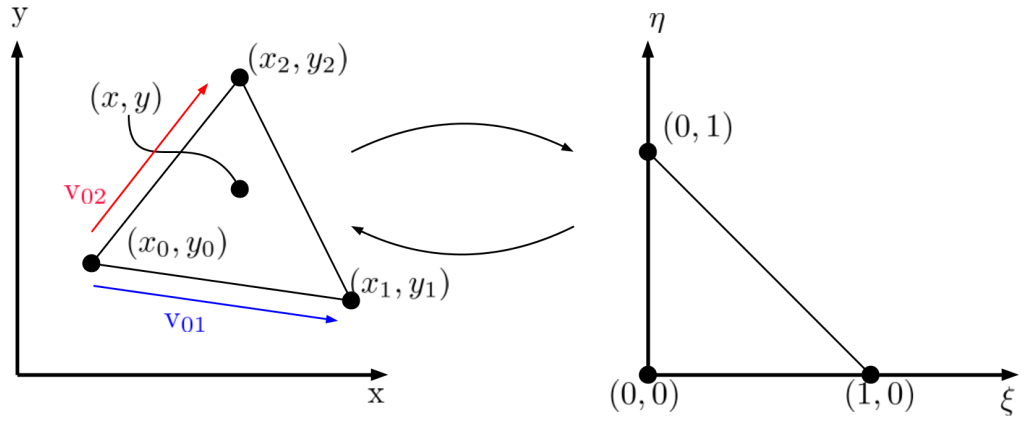

Figure 1: Mapping of a 2D triangle to a reference triangle in natural coordinates.

The linear basis functions for the reference triangle are

$$N_0(\xi, \eta) = 1 - \xi - \eta$$
$$N_1(\xi, \eta) = \xi$$
$$N_2(\xi, \eta) = \eta.$$

From these functions we can interpolate the point (x, y) with the following

$$x = N_0 x_0 + N_1 x_1 + N_2 x_2$$
$$y = N_0 y_0 + N_1 y_1 + N_2 y_2$$

We can now express x and y as functions of ξ and η by substituting the expressions for N_0, N_1 and N_2 into the expressions for x and y

$$\begin{aligned} x &= (1 - \xi - \eta)x_0 + (\xi)x_1 + (\eta)x_2 \\ &= x_0 - \xi x_0 - \eta x_0 + \xi x_1 + \eta x_2 \\ &= x_0 + (x_1 - x_0)\xi + (x_2 - x_0)\eta \end{aligned}$$

and

$$\begin{aligned} y &= (1-\xi-\eta)y_0 + (\xi)y_1 + (\eta)y_2 \\ &= y_0 - \xi y_0 - \eta y_0 + \xi y_1 + \eta y_2 \\ &= y_0 + (y_1 - y_0)\xi + (y_2 - y_0)\eta \end{aligned}$$

In terms of the vectors from vertex 0 to the other two vertices (refer to Figure 1) we can write this as

$$x = x_0 + \mathrm{v}_{01x}\xi + \mathrm{v}_{02x}\eta \tag{1}$$

$$y = y_0 + \mathrm{v}_{01y}\xi + \mathrm{v}_{02y}\eta \tag{2}$$

which is in the form of a linear transformation and from which we can determine the very important Jacobian matrix

$$\mathbf{J} = \begin{bmatrix} \frac{dx}{d\xi} & \frac{dx}{d\eta} \\ \frac{dy}{d\xi} & \frac{dy}{d\eta} \end{bmatrix} = \begin{bmatrix} \mathrm{v}_{01x} & \mathrm{v}_{02x} \\ \mathrm{v}_{01y} & \mathrm{v}_{02y} \end{bmatrix} = \begin{bmatrix} (x_1 - x_0) & (x_2 - x_0) \\ (y_1 - y_0) & (y_2 - y_0) \end{bmatrix}. \tag{3}$$

The first application of the Jacobian will be for the integration of the trial or basis function in the finite element method. For simplicity let us consider the integration of a function over x and y which can be transformed to an integration of the linear transformation of function f, i.e. function g, over ξ and η using fundamental linear algebra. This integration is then

$$\int\int f(x,y).dx.dy = \int\int g(\xi,\eta).|J|.d\xi.d\eta$$

where $|J|$ is the determinant of the Jacobian. This integration can then easily be done either analytically or by using a quadrature rule. For the reference triangle this can easily be done using the method of undetermined coefficients as detailed in appendix A.

We need to define one more item that is related to the finite element method and that is the derivative of the basis functions, $\nabla N_i(\xi,\eta)$, which can be developed by noting that

$$\begin{aligned} \frac{\partial N_i}{\partial \xi} &= \frac{\partial N_i}{\partial x}\cdot\frac{\partial x}{\partial \xi} + \frac{\partial N_i}{\partial y}\cdot\frac{\partial y}{\partial \xi} \\ \frac{\partial N_i}{\partial \eta} &= \frac{\partial N_i}{\partial x}\cdot\frac{\partial x}{\partial \eta} + \frac{\partial N_i}{\partial y}\cdot\frac{\partial y}{\partial \eta} \end{aligned}$$

which can be written as

$$\begin{bmatrix} \frac{\partial N_i}{\partial \xi} \\ \frac{\partial N_i}{\partial \eta} \end{bmatrix} = \begin{bmatrix} \frac{\partial x}{\partial \xi} & \frac{\partial y}{\partial \xi} \\ \frac{\partial x}{\partial \eta} & \frac{\partial y}{\partial \eta} \end{bmatrix} \begin{bmatrix} \frac{\partial N_i}{\partial x} \\ \frac{\partial N_i}{\partial y} \end{bmatrix}$$

$$\therefore \begin{bmatrix} \frac{\partial N_i}{\partial \xi} \\ \frac{\partial N_i}{\partial \eta} \end{bmatrix} = \mathbf{J}^T \begin{bmatrix} \frac{\partial N_i}{\partial x} \\ \frac{\partial N_i}{\partial y} \end{bmatrix}.$$

Now we can invert $\mathbf{J}^T$ to get

$$\begin{bmatrix} \frac{\partial N_i}{\partial x} \\ \frac{\partial N_i}{\partial y} \end{bmatrix} = (\mathbf{J}^T)^{-1} \begin{bmatrix} \frac{\partial N_i}{\partial \xi} \\ \frac{\partial N_i}{\partial \eta} \end{bmatrix} \tag{4}$$

and since the inverse of a 2×2 matrix is given by

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix}^{-1} = \frac{1}{ad - bc} \begin{bmatrix} d & -b \\ -c & a \end{bmatrix}$$

we have

$$(\mathbf{J}^T)^{-1} = \begin{bmatrix} \frac{\partial x}{\partial \xi} & \frac{\partial y}{\partial \xi} \\ \frac{\partial x}{\partial \eta} & \frac{\partial y}{\partial \eta} \end{bmatrix}^{-1} = \frac{1}{|J|} \begin{bmatrix} \frac{\partial y}{\partial \eta} & -\frac{\partial y}{\partial \xi} \\ -\frac{\partial x}{\partial \eta} & \frac{\partial x}{\partial \xi} \end{bmatrix} \tag{5}$$

2 Piecewise linear shape functions on a 2D polygon

The development of the methodology as applied to a 2D triangle has direct application when applied to a polygon since a polygon most likely presents more complexity than classical reference elements like quadrilaterals and therefore using triangles as subsets of polygons overcomes this complexity. The use of subset triangles for the representation of a polygon was presented by Bailey & Adams in [1] the same authors which subsequently studied bi-linear basis functions [2]. From the latter paper it is this author's judgement that bi-linear basis functions offer little benefit over their linear counterparts and we will therefore pursue the linear methods presented in [1]. The basis functions for each vertex of a polygon are of the form

$$P_i(x, y) = N_i(x, y) + \beta_i N_c(x, y) \tag{6}$$

where the functions N_i and N_c are the standard linear functions defined on triangles. The subscripts i and c refer to the vertices i and center of the polygon, respectively. The β_i value is a weighting constant defined such that

$$\begin{bmatrix} x \\ y \end{bmatrix}_c = \sum_{s=0}^{N_s} \beta_s \begin{bmatrix} x \\ y \end{bmatrix}_{s,avg} . \tag{7}$$

Naturally it follows that $\beta_s = \dfrac{1}{N_s}$ where N_s is the amount of sides. $[x\ y]_{s,avg}$ is the average coordinate of the two vertices of a side. An example basis function is shown in Figure 2. It should be noted that a single basis function now requires integration on each of the sub-triangles of the polygon instead of just a single one.

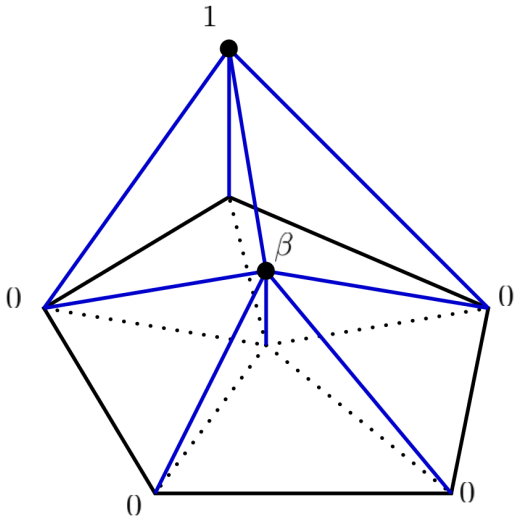

Figure 2: Basis function on a 2D polygon.

It is hard to visualize that this approach leads to an equivalent representation but with a few tests one can establish that this is equivalent. Some aspects of the paper presenting this method [1] that are not intuitive is the explanation that integration is now over all sides. Also, the paper does not explicitly state that the cell centroid never features as an unknown in the solution. The customary assembly of the matrix in triangle or quadrilateral based meshes is to assemble cell-by-cell with an inner loop over the DOF of the cell. This approach is essentially the same but the paper states that integration is per side without saying that each DOF (except the cell center) of each side is also an inner loop of this integration.

The method is versatile enough to applied to triangles and quadrilaterals where examples of the shape functions are shown in Figure

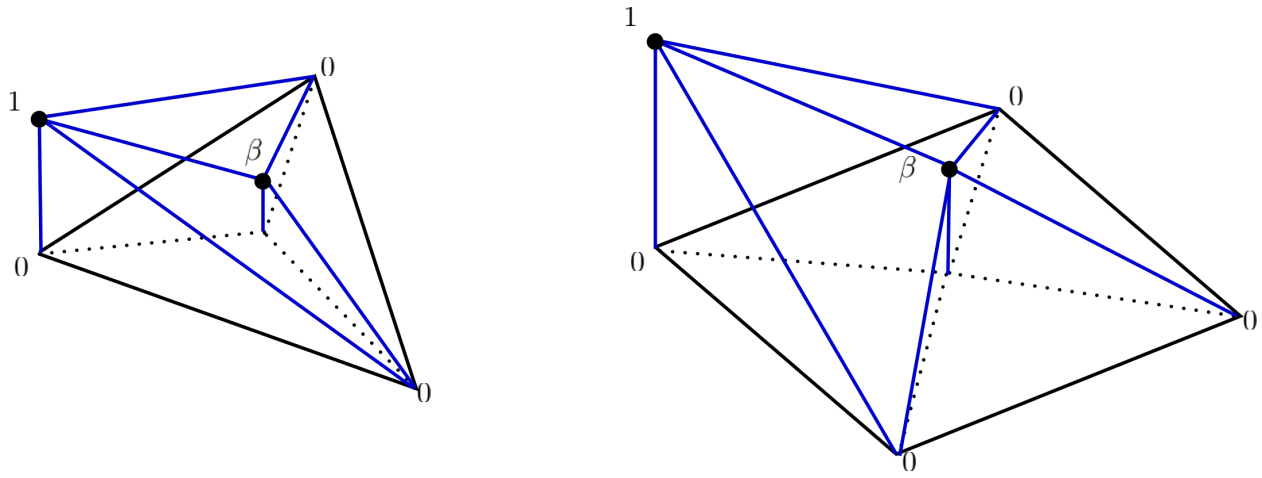

Figure 3: Basis functions on a triangle and on a quadrilateral.

3 Volume Integrals

IntV_gradShapeI_gradShapeJ[i][j]
IntV_shapeI_gradshapeJ[i][j]
IntV_shapeI_shapeJ[i][j]
IntV_shapeI[i]

The finite element method requires the integration of the shape functions over the volume and surface of the cell. These integrations can easily be made analytically but for the purpose of being easy to follow we will revert to integration using a Gaussian quadrature. Let us discuss the assembly of the volume integrals, which for 2D polygons are essentially integrals over area multiplied by a unitary height. Before we do so we have to understand the assembly of the integrals in a triangle-by-triangle fashion. Volume integrals therefore take the form

$$\begin{aligned}\int_V P_i P_j.dV &= \int_V \Big[N_i + \beta_i N_c\Big]\Big[N_j + \beta_j N_c\Big].dV \\ &= \sum_{tris} \int_{V_{tri}} \Big[N_i + \beta_i N_c\Big]\Big[N_j + \beta_j N_c\Big].dV_{tri}\end{aligned} \tag{8}$$

Now let

$$F(x,y) = \Big[N_i + \beta_i N_c\Big]\Big[N_j + \beta_j N_c\Big]$$

With this definition we now have the more general form of a function evaluated triangle-by-triangle which is represented by

$$\int_V P_i P_j.dV = \int_V F(x,y).dV = \sum\nolimits_{tris} \int_{V_{tri}} F(x,y).dV_{tri} \tag{9}$$

for which we can first transform the integral over cartesian coordinate version of a triangle (V_{tri}) to an integral over the natural coordinates version (V_τ) as

$$\int_{V_{tri}} F(x,y).dV_{tri} = \int_{V_\tau} F(\xi,\eta).dV_\tau.|J|_{tri} \tag{10}$$

and then apply an appropriate Gaussian quadtrature for the volume integral

$$\int_{V_\tau} F(\xi,\eta).dV_\tau.|J|_{tri} = \sum_q^Q F((\xi,\eta)_q).w_q.|J|_{tri} \tag{11}$$

where w_q is the quadrature weight associated with the quadtrature abscissa q, a unique comination of ξ and η. These weights and abscissae are tabulated in Appendix A.

4 Surface Integrals

`IntS_shapeI_shapeJ[f][i][j]`
`IntS_shapeI[f][i]`
`IntS_shapeI_gradshapeJ[f][i][j]`

The same form of the volume integrals can be encountered on surfaces of the cell (in this case an edge). To this end we can use the convention depicted in Figure 4. With the first leg of the triangle defined along ξ the integral along the edge of the triangle is always an integral from 0 to 1 along the natural coordinates.

$$v_0 = \text{Edge vertex 0}$$
$$v_1 = \text{Edge vertex 1}$$
$$v_2 = \text{Cell center}$$

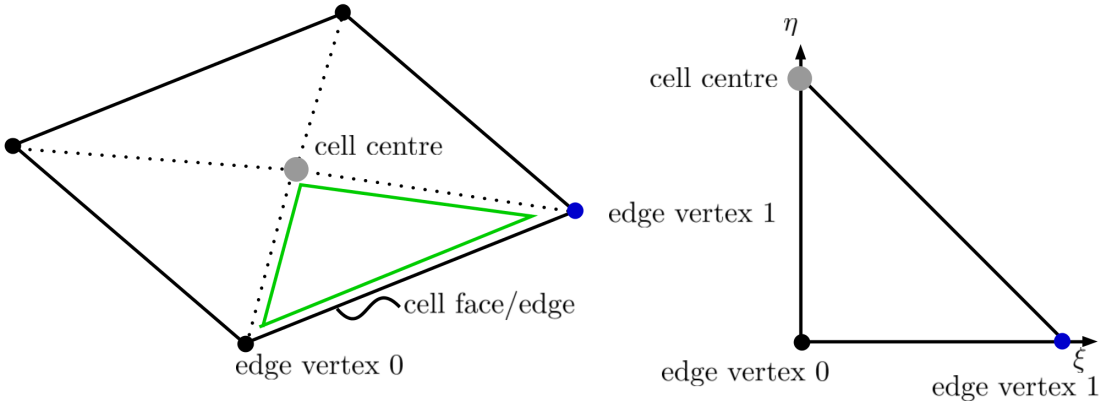

Figure 4: Orientation of a triangle on a cell.

Surface integrals can now take the simple form of the volume integrals without any special treatment

$$\int_{side} F(x,y).dA_{side} = \sum_{q}^{Q} F(\xi_q,\eta).w_q.|J|_{surf} \tag{12}$$

The determinant on the surface, $|J|_{surf}$, is simply the length of the edge in cartesian coordinates. Quadrature weights and abscissae for integration are depicted in Appendix B.

5 Coding implementation

5.1 5.1 Constructor 00_constrdestr.cc

In the constructor we receive the basic cell information (i.e. information about vertices and edges), a reference to the grid (holding actual vertex values) and the discretization method (holding quadrature information).

```
content...
```

```
PolygonFEView(chi_mesh::CellPolygon* poly_cell,
              chi_mesh::MeshContinuum* grid,
              CHI_DISCRETIZATION_PWL *discretization)
```

- We immediately make reference copies of the quadrature sets and obtain the cell center.
- We then construct each triangle of the polygon by looping over the edges of the cell
  - For each edge we instantiate a new data structure, FEside_data2d , which holds the triangle data.
  - For each edge we form the v_{01} leg of the triangle by the cell edge. And the v_{02} leg as the edge from the first vertex of the edge to the cell centre.
  - Since the determinant of the area is easy to compute we do it here.
  - Since the length of the v_{01} leg is unity in natural coordinates, the edge determinant (more generally surface determinant) is simply the length (L_2-norm) of the v_{01} leg.
  - We then compute the Jacobian, Jacobian inverse and the Jacobian-transpose inverse
- In order to determine if a given cell DOF is associated with vertex 0 or vertex 1 of a given triangle we have to develop a node-to-side mapping.
- Lastly a useful mapping is developed where, given a side and side DOF index, we can obtain the associated cell DOF. This is useful when assembling matrices.

5.2 Precomputing the integrals

The precompute function, also called the phase of "computing cell matrices", is a function which precomputes the quadrature rule based integrals. For its first phase it computes the values of the different

shape functions on every triangle and every quadrature point. To this end the code relies on the following functions:

Defined in 02_compcellmat.cc

```
double PreShape(int s, int i, int qpoint_index, bool on_surface = false);
double PreGradShape_x(int s, int i, int qpoint_index);
double PreGradShape_y(int s, int i, int qpoint_index);
```

These functions use the developed FEside_data2d data structure to evaluate the side shape functions from the reference triangle function.

As an example consider PreGradShape_x :

```
double PolygonFEView::PreGradShape_x(int s, int i, int qpoint_index)
{
  int index = node_to_side_map[i][s];
  double value = 0;
  if (index==0)
  {

    value = sides[s]->JTinv.GetIJ(0,0)*-1.0 +
            sides[s]->JTinv.GetIJ(0,1)*-1.0;
  }
  if (index==1)
  {

    value = sides[s]->JTinv.GetIJ(0,0)*1.0 +
            sides[s]->JTinv.GetIJ(0,1)*0.0;
  }

  value += beta*(sides[s]->JTinv.GetIJ(0,0)*0.0 +
                 sides[s]->JTinv.GetIJ(0,1)*1.0);


  return value;
}
```

This trail culminates in the following portion of the PreCompute phase:

```
for (int s=0; s<num_of_subtris; s++)
{
  for (int i=0; i<dofs; i++)
  {
    FEqp_data2d* pernode_data = new FEqp_data2d;
    for (int q=0; q<vol_quadrature->qpoints.size(); q++)
    {
      pernode_data->shape_qp.push_back(PreShape(s, i, q));
      pernode_data->gradshapex_qp.push_back(PreGradShape_x(s, i, q));
      pernode_data->gradshapey_qp.push_back(PreGradShape_y(s, i, q));
    }//for qp

    for (int q=0; q<surf_quadrature->abscissae.size(); q++)
    {
      //printf("%g\n",PreShape(s,i,q,ON_SURFACE));
      pernode_data->shape_qp_surf.push_back(PreShape(s,i,q,ON_SURFACE));
```

```
    }
    sides[s]->qp_data.push_back(pernode_data);
  }//for dof
}//for side
```

With the values of the shape functions evaluated at the quadrature points the second phase of the PreCompute function assembles the integrals as if we were assembling a matrix by using the quadrature points. It has many facets and a number of utility functions but should be intuitive to follow.

6 Using interpolation

Two utility functions are available for use by field function interpolators:

Defined in 01_xy_shapefuncs.cc

```
double Shape_xy(int i, chi_mesh::Vector xyz);
chi_mesh::Vector GradShape_xy(int i, chi_mesh::Vector xyz);
```

These functions are used with cartesian coordinates and returns the values of the shape functions in the cartesian reference frames.

Appendix A Quadrature rule for integration of triangle space

We seek an integral of a function in triangle space T_{sp} in the form

$$\int\int_{T_{sp}} f(x,y).dx.dy = \sum_{i=0}^{N-1} w_i f(x_i, y_i).$$

Furthermore we know that in the finite element method with only linear shape functions we will at most have polynomials of degree 2 therefore we can devise a set of test functions

$$\begin{aligned}
f(x,y) = 1 \quad & \int_0^1 \int_0^{1-y} 1.dx.dy = \frac{1}{2} = \sum_{i=0}^{N-1} w_i \\
f(x,y) = x \quad & \int_0^1 \int_0^{1-y} x.dx.dy = \frac{1}{6} = \sum_{i=0}^{N-1} w_i x_i \\
f(x,y) = y \quad & \int_0^1 \int_0^{1-y} y.dx.dy = \frac{1}{6} = \sum_{i=0}^{N-1} w_i y_i \\
f(x,y) = xy \quad & \int_0^1 \int_0^{1-y} xy.dx.dy = \frac{1}{24} = \sum_{i=0}^{N-1} w_i x_i y_i \\
f(x,y) = x^2 \quad & \int_0^1 \int_0^{1-y} x^2.dx.dy = \frac{1}{12} = \sum_{i=0}^{N-1} w_i x_i^2 \\
f(x,y) = y^2 \quad & \int_0^1 \int_0^{1-y} y^2.dx.dy = \frac{1}{12} = \sum_{i=0}^{N-1} w_i y_i^2
\end{aligned}$$

With $N = 3$ a symmetric solution is obtained with

$$\begin{aligned}
w_i &= \frac{1}{6} \\
x_0, y_0 &= (\frac{1}{6}, \frac{1}{6}) \\
x_1, y_1 &= (\frac{4}{6}, \frac{1}{6}) \\
x_2, y_2 &= (\frac{1}{6}, \frac{4}{6})
\end{aligned}$$

which is not a unique solution.

Appendix B Quadrature rule for integration of triangle edge

Along the edge of a triangle the integration reverts to a one dimensional integral

$$\int_S F(x,y).dA \rightarrow \int_e F(x,y).ds = \int_e F(\xi,\eta).ds.|J| \tag{13}$$

In natural coordinates we will mostly deal with the following integral

$$\begin{aligned} \int_0^1 F(\xi,\eta).d\xi &= \frac{1}{2}\int_{-1}^{+1} F(y,\eta).dy \\ &= \frac{1}{2}\sum_{q=0}^{1} w_q F(y_q,\eta) \end{aligned} \tag{14}$$

where $y = \frac{1}{2}\xi + \frac{1}{2}$ and for which the quadrature rule is

$$\begin{aligned} y_0 &= +\frac{1}{\sqrt{3}} & w_0 &= 1 \\ y_1 &= -\frac{1}{\sqrt{3}} & w_1 &= 1 \end{aligned}$$

Now when translated back to natural coordinates in the form

$$\int_0^1 F(\xi,\eta).d\xi = \sum_{q=0}^{1} w'_q F(\xi_q,\eta)$$

we get the quadrature rule

$$\begin{aligned} \xi_0 &= +\frac{1}{2\sqrt{3}} + \frac{1}{2} & w_0 &= \frac{1}{2} \\ \xi_1 &= -\frac{1}{2\sqrt{3}} + \frac{1}{2} & w_1 &= \frac{1}{2} \end{aligned}$$

References

[1] Bailey T.S., Chang J.H., Adams M.L., *A Piecewise Linear Discontinous Finite Element spatial discretization of the transport equation in 2D Cylindrical Geometry*, 2009 International Conference on Advances in Mathematics, Computational Methods, and Reactor Physics, 2008.

[2] Bailey T.S., Warsa J.S., Chang J.H., Adams M.L., *A Piecewise Bi-linear Discontinous Finite Element spatial discretization of the S_n transport equation*, International Conference on Mathematics and Computational Methods Applied to Nuclear Science and Engineering, 2011.