Boundary Value Problems: Constructing the Element Stiffness Matrix and Forcing Vector

Now that we know how to assemble the matrix system resulting from Galerkin's method, one question remains: how does one construct the element stiffness matrix and forcing vector? To do so, we take advantage of three things:

(i) The isoparametric concept,
(ii) Bézier extraction, and
(iii) Quadrature.

To ground our discussion, let $\Omega_h^e \subseteq \Omega_h$ denote an arbitrary element in physical space. Then we are interested in computing the following:

$$k_{ab}^e = \int_{\Omega_h^e} K \, \vec{\nabla} N_a^e \cdot \vec{\nabla} N_b^e \, d\Omega_h^e \qquad a,b = 1,\dots,n_{loc}$$

$$+ \int_{\Gamma_R \cap \Gamma_h^e} \beta \, N_a^e \, N_b^e \, d\Gamma^e \qquad \Gamma_h^e \hookrightarrow \partial\Omega_h^e$$

$$f_a^e = \int_{\Omega_h^e} N_a^e \, f \, d\Omega_h^e \qquad a = 1,\dots,n_{loc}$$

$$+ \int_{\Gamma_N \cap \Gamma_h^e} N_a^e \, h \, d\Gamma^e + \int_{\Gamma_R \cap \Gamma_h^e} \beta \, N_a^e \, u_R \, d\Gamma^e \qquad \Gamma_h^e \hookrightarrow \partial\Omega_h^e$$

We now pullback to the parent element $\hat{\Omega}_h$ using the inverse mapping $(\vec{x}^e)^{-1} : \Omega_h^e \to \hat{\Omega}_h$. In what follows, we need the following objects:

$\underline{\underline{J}}$: Jacobian matrix, defined as $\underline{\underline{J}} := \vec{x}_{,\hat{\xi}} = \begin{bmatrix} \frac{\partial x}{\partial \xi_1} & \frac{\partial x}{\partial \xi_2} \\ \frac{\partial y}{\partial \xi_1} & \frac{\partial y}{\partial \xi_2} \end{bmatrix}$

j : Determinant, or Jacobian determinant, defined as $j := \det(\underline{\underline{J}})$

$\hat{\underline{n}}$: Outward facing normal to $\hat{\Omega}_h$ along $\hat{\Gamma} = \partial\hat{\Omega}_h$ (unit)

$\hat{\underline{t}}$: Tangent along $\hat{\Gamma} = \partial\hat{\Omega}_h$ (unit)

j_Γ : Surface determinant, defined as $j_\Gamma := |\underline{\underline{J}}\,\hat{\underline{t}}|$ along $\hat{\Gamma}$

With the above defined, we may now pull all of our integrals back to the parent element $\hat{\Omega}_h$.

$$\int_{\Omega_l^e} K \vec{\nabla} N_a^e \cdot \vec{\nabla} N_b^e \, d\Omega_l^e = \int_{\tilde{\Omega}_l} K(\vec{x}^e(\vec{\xi})) \left(\vec{\nabla}_{\vec{x}} N_a^e(\vec{x}^e(\vec{\xi}))\right) \cdot \left(\vec{\nabla}_{\vec{x}} N_b^e(\vec{x}^e(\vec{\xi}))\right) \frac{d\Omega_l^e}{d\tilde{\Omega}_l} \, d\tilde{\Omega}_l$$

j

$$\int_{\Gamma_R \cap \Gamma^e} \mathcal{B} N_a^e N_b^e \, d\Gamma^e = \int_{\tilde{\Gamma}_{R,e}} \mathcal{B}(\vec{x}^e(\vec{\xi})) N_a^e(\vec{x}^e(\vec{\xi})) N_b^e(\vec{x}^e(\vec{\xi})) \frac{d\Gamma^e}{d\tilde{\Gamma}} \, d\tilde{\Gamma}$$

j_Γ

$$\int_{\Omega_l^e} N_a^e f \, d\Omega_l^e = \int_{\tilde{\Omega}_l} N_a^e(\vec{x}^e(\vec{\xi})) \, f(\vec{x}^e(\vec{\xi})) \frac{d\Omega_l^e}{d\tilde{\Omega}_l} \, d\tilde{\Omega}_l$$

j

$$\int_{\Gamma_N \cap \Gamma^e} N_a^e h \, d\Gamma^e = \int_{\tilde{\Gamma}_{N,e}} N_a^e(\vec{x}^e(\vec{\xi})) h(\vec{x}^e(\vec{\xi})) \frac{d\Gamma^e}{d\tilde{\Gamma}} \, d\tilde{\Gamma}$$

j_Γ

$$\int_{\Gamma_R \cap \Gamma^e} \mathcal{B} N_a^e u_R \, d\Gamma^e = \int_{\tilde{\Gamma}_{R,e}} \mathcal{B}(\vec{x}^e(\vec{\xi})) N_a^e(\vec{x}^e(\vec{\xi})) u_R(\vec{x}^e(\vec{\xi})) \frac{d\Gamma^e}{d\tilde{\Gamma}} \, d\tilde{\Gamma}$$

j_Γ

where we have defined:

$$\tilde{\Gamma}_{N,e} = \vec{x}^{e\,-1}(\Gamma_N \cap \Gamma^e) \quad \text{(preimage of Neumann boundary)}$$

$$\tilde{\Gamma}_{R,e} = \vec{x}^{e\,-1}(\Gamma_R \cap \Gamma^e) \quad \text{(preimage of Robin boundary)}$$

In what follows, we will omit dependencies on $\vec{x}^e(\vec{\xi})$ where it is obvious. For example, we write:

$$k_{ab}^e = \int_{\tilde{\Omega}_l} K \vec{\nabla}_{\vec{x}} N_a^e \cdot \vec{\nabla}_{\vec{x}} N_b^e \; j \, d\tilde{\Omega}_l + \int_{\tilde{\Gamma}_{R,e}} \mathcal{B} N_a^e N_b^e \, j_\Gamma \, d\tilde{\Gamma}$$

$$f_a^e = \int_{\tilde{\Omega}_l} N_a^e f \, j \, d\tilde{\Omega}_l + \int_{\tilde{\Gamma}_{N,e}} N_a^e h \, j_\Gamma \, d\tilde{\Gamma} + \int_{\tilde{\Gamma}_{R,e}} \mathcal{B} N_a^e u_R \, j_\Gamma \, d\tilde{\Gamma}$$

At this point, we have two questions:

(i) How do we compute basis functions and their derivatives?

(ii) How do we evaluate the parent element integrals?

Let us first deal with (i). Recall that we have:

$$\underline{N}^e(\vec{x}^e(\vec{\xi})) = \underline{\underline{C}}^e \, \underline{B}(\vec{\xi})$$

- $\underline{N}^e$: Vector of B-spline Basis Functions
- $\underline{\underline{C}}^e$: Extraction Operator
- $\underline{B}$: Vector of Bernstein Basis Functions Over the Parent Element

in the B-Spline setting. Consequently, if we evaluate the Bernstein basis, we can easily find the B-spline basis by multiplying through by the extraction operator. In the NURBS setting, we simply divide through by the weighting function as well. In matrix form:

$$\underline{R}^e(\vec{x}^e(\vec{\xi})) = \underline{\underline{W}}^e \frac{\underline{N}^e(\vec{x}^e(\vec{\xi}))}{w^e(\vec{\xi})}$$

($w^e(\vec{\xi})$: Weighting Function)

$$\text{Vector of NURBS Basis Functions} = \underline{\underline{W}}^e \underline{\underline{C}}^e \frac{\underline{B}(\vec{\xi})}{w^e(\vec{\xi})}$$

where:

$$\underline{\underline{W}}^e = \begin{bmatrix} w_1^e & & & \\ & w_2^e & & \\ & & \ddots & \\ & & & w_{n_{loc}}^e \end{bmatrix}$$

To evaluate derivatives, we employ the chain rule. For instance:

$$\vec{\nabla}_{\vec{x}} N_a^e = \left(\frac{\partial \vec{\xi}}{\partial \vec{x}}\right)^T \vec{\nabla}_{\vec{\xi}} N_a^e = \left(\left(\underline{\underline{J}}\right)^{-1}\right)^T \vec{\nabla}_{\vec{\xi}} N_a^e$$

$$\vec{\nabla}_{\vec{x}} N_b^e = \left(\frac{\partial \vec{\xi}}{\partial \vec{x}}\right)^T \vec{\nabla}_{\vec{\xi}} N_b^e = \left(\left(\underline{\underline{J}}\right)^{-1}\right)^T \vec{\nabla}_{\vec{\xi}} N_b^e$$

To evaluate the derivatives with respect to the parent element, we again lean on Bézier extraction. Then:

$$\frac{\partial \underline{R}^e}{\partial \xi_i} = \underline{\underline{W}}^e \underline{\underline{C}}^e \frac{\partial}{\partial \xi_i}\left(\frac{\underline{B}^e(\vec{\xi})}{w^e(\vec{\xi})}\right)$$

$$= \underline{\underline{W}}^e \underline{\underline{C}}^e \left(\frac{1}{w^e(\vec{\xi})} \frac{\partial \underline{B}^e(\vec{\xi})}{\partial \xi_i} - \frac{\partial w^e(\vec{\xi})}{\partial \xi_i} \frac{\underline{B}^e(\vec{\xi})}{(w^e(\vec{\xi}))^2}\right)$$

and:

$$\frac{\partial \underline{R}^e}{\partial x} = \frac{\partial \underline{R}^e}{\partial \underline{\hat{\xi}}_1}\frac{\partial \underline{\hat{\xi}}_1}{\partial x} + \frac{\partial \underline{R}^e}{\partial \underline{\hat{\xi}}_2}\frac{\partial \underline{\hat{\xi}}_2}{\partial x}$$

$$\frac{\partial \underline{R}^e}{\partial y} = \frac{\partial \underline{R}^e}{\partial \underline{\hat{\xi}}_1}\frac{\partial \underline{\hat{\xi}}_1}{\partial y} + \frac{\partial \underline{R}^e}{\partial \underline{\hat{\xi}}_2}\frac{\partial \underline{\hat{\xi}}_2}{\partial y}$$

As we shall see later, we will encode these operations in a shape function routine.

We finally turn to (ii), evaluation of the parent element integrals. We will employ Gaussian quadrature to numerically approximate the integrals. Gaussian quadrature typically is assumed to take place over the domain $[-1,1]$, so we map the standard rules to the interval $[0,1]$ so we can employ our standard parent element.

The n-point Gauss quadrature rule takes the form:

$$\int_0^1 f(\underline{\hat{\xi}})\, d\underline{\hat{\xi}} \approx \sum_{q=1}^{n_q} \hat{w}_q\, f(\underline{\hat{\xi}}_q)$$

$\hat{w}_q$: q^{th} quadrature weight; $\underline{\hat{\xi}}_q$: q^{th} quadrature point

Remark: Not NURBS weights!

where:

Number of Points, n_q	Points, $\underline{\hat{\xi}}_q$	Weights, $\hat{w}_q$
1	$0.5 \to 1/2$	1
2	$1/2 \pm 1/2\sqrt{1/3}$	1/2
3	$1/2$	$4/9$
	$1/2 \pm 1/2\sqrt{3/5}$	$5/18$
4	$1/2 \pm 1/2\sqrt{\frac{3}{7} - \frac{2}{7}\sqrt{\frac{6}{5}}}$	$1/2\left(\frac{18+\sqrt{30}}{36}\right)$
	$1/2 \pm 1/2\sqrt{\frac{3}{7} + \frac{2}{7}\sqrt{\frac{6}{5}}}$	$1/2\left(\frac{18-\sqrt{30}}{36}\right)$

As a general rule, $(p+1)$-point Gauss quadrature should be used when employing NURBS basis functions of degree p.

In two-spatial dimensions, a tensor-product Gauss quadrature rule is employed:

$$\int_{\hat{\Omega}} f(\underline{\hat{\xi}})\, d\hat{\Omega} \approx \sum_{q_1=1}^{n_q}\sum_{q_2=1}^{n_q} \hat{w}_{q_1}\hat{w}_{q_2}\, f(\hat{\xi}_{q_1}, \hat{\xi}_{q_2})$$

and again, as a general rule, $(p+1)$-point Gauss quadrature should be used in each direction for a degree-p NURBS basis.

Note that, when combining the isoparametric concept with quadrature, we have:

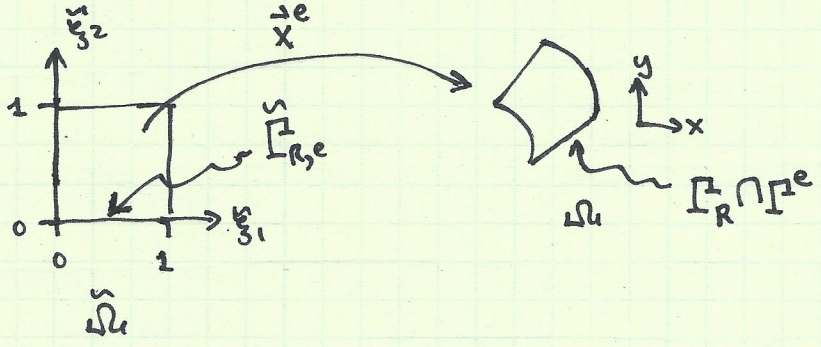
$$\int_{\Omega^e} K \vec{\nabla} N_a^e \cdot \vec{\nabla} N_b^e \, d\Omega^e \approx \sum_{q_1=1}^{n_q} \sum_{q_2=1}^{n_q} K(\vec{x}^e(\hat{\xi}_{q_1}, \hat{\xi}_{q_2})) * \left(\vec{\nabla}_{\vec{x}} N_a^e(\vec{x}^e(\hat{\xi}_{q_1}, \hat{\xi}_{q_2}))\right) \cdot \left(\vec{\nabla}_{\vec{x}} N_b^e(\vec{x}^e(\hat{\xi}_{q_1}, \hat{\xi}_{q_2}))\right) * \hat{w}_{q_1} \hat{w}_{q_2} j(\hat{\xi}_{q_1}, \hat{\xi}_{q_2})$$

Thus, to evaluate the integral, we have to perform a loop over the quadrature points and then, for each quadrature point, we need to evaluate K, $\vec{\nabla}_x N_a^e$, $\vec{\nabla}_x N_b^e$, and j. We do this in our shape function routine which uses Bézier extraction to compute NURBS quantities.

On the element boundaries, we perform analogous operations. Suppose that:

$$\hat{\Gamma}_{R,e} := \{ \hat{\xi} = (t, 0) : t \in [0,1] \}$$

Visually:



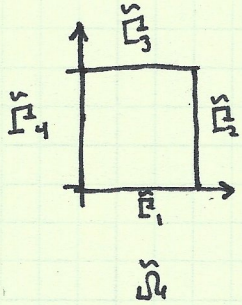
Then:

$$\int_{\Gamma_R \cap \Gamma^e} \beta N_a^e N_b^e \, d\Gamma^e \approx \sum_{q=1}^{n_q} \beta(\vec{x}^e(\hat{\xi}_q, 0)) N_a^e(\vec{x}^e(\hat{\xi}_q, 0)) N_b^e(\vec{x}^e(\hat{\xi}_q, 0)) * \hat{w}_q \, j_\Gamma(\hat{\xi}_q, 0)$$

On $\hat{\Gamma}_{R,e}$, the tangent vector is $\underline{t} = \{1, 0\}^T$, so we see:

$$j_\Gamma(\hat{\xi}_q, 0) = \left| \underline{\underline{J}}(\hat{\xi}_q, 0) \underline{t} \right| = \left| \underline{\underline{J}}(\hat{\xi}_q, 0) \begin{bmatrix} 1 \\ 0 \end{bmatrix} \right|$$

We have analogous expressions for boundary integrals over the other three sides of $\hat{\Gamma}$. To aid in implementation, we can create two arrays, Neumann and Robin, which identify if a given side if a Bézier element belongs to either the Neumann and/or Robin boundary.



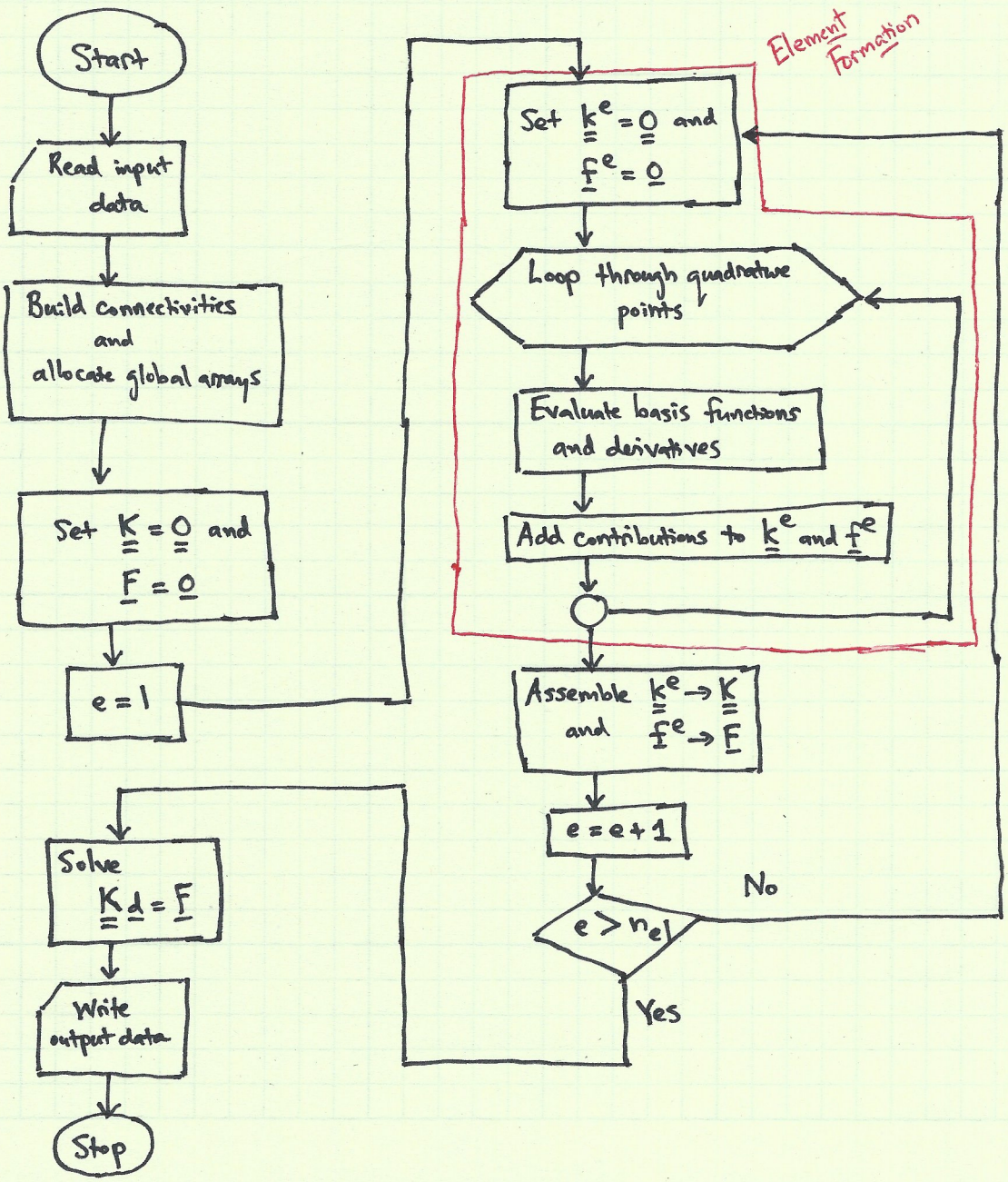
$$\text{Neumann}(a,e) = \begin{cases} 1 & \text{if } \vec{x}^e(\tilde{\Gamma}_a) \subseteq \Gamma_N \\ 0 & \text{otherwise} \end{cases}$$

(Side: a, Element: e)

$$\text{Robin}(a,e) = \begin{cases} 1 & \text{if } \vec{x}^e(\tilde{\Gamma}_a) \subseteq \Gamma_R \\ 0 & \text{otherwise} \end{cases}$$

The above arrays may be constructed as a preprocessing step.

With the aforementioned terminology defined, we can now present pseudo code for construction of the element stiffness matrix and forcing vector array. Before doing so, we present a flowchart giving the overall formation and assembly process.



We are currently concerned with the element formation portion of the overall formation and assembly process. As illustrated above, it involves a loop over quadrature points and evaluation of basis functions and derivatives at these quadrature points. In fact, we have three loops, one over the element interior, one over the Neumann boundary, and one over the Robin boundary. This is made obvious in the algorithm below.

Initialize
$\underline{\underline{k}}^e = 0$
$\underline{f}^e = 0$

Loop over Element Interiors

for $q_1 = 1, \ldots, n_q$
  for $q_2 = 1, \ldots, n_q$

    Call Shape Function $(e, \hat{\xi}_{q_1}, \hat{\xi}_{q_2})$ which computes the NURBS basis functions R_a^e, their derivatives $\partial R_a^e / \partial x$ and $\partial R_a^e / \partial y$, the location $\vec{x}$, and the Jacobian matrix $\underline{\underline{J}}$ at $\vec{x}^e(\hat{\xi}_{q_1}, \hat{\xi}_{q_2})$

    Set $j = \det(\underline{\underline{J}})$
    Set $K_{loc} = K(\vec{x})$
    Set $f_{loc} = f(\vec{x})$

    for $a = 1, \ldots, n_{loc}$
      for $b = 1, \ldots, n_{loc}$

        Update $k_{ab}^e = k_{ab}^e + K_{loc} * ((\vec{\nabla}_{\vec{x}} R_a^e) \cdot (\vec{\nabla}_{\vec{x}} R_b^e)) * \hat{w}_{q_1} * \hat{w}_{q_2} * j$

      endloop

      Update $f_a^e = f_a^e + R_a^e * f_{loc} * \hat{w}_{q_1} * \hat{w}_{q_2} * j$

    endloop

  endloop

endloop

for side $= 1, \ldots, 4$

  if Neumann(side, e) $= 1$ or Robin(side, e) $= 1$
    for $q = 1, \ldots, n_q$

      Set $\hat{\xi}_1 = \begin{cases} \hat{\xi}_q & \text{if side} = 1,3 \\ 1 & \text{if side} = 2 \\ 0 & \text{if side} = 4 \end{cases}$

      Set $\hat{\xi}_2 = \begin{cases} \hat{\xi}_q & \text{if side} = 2,4 \\ 1 & \text{if side} = 3 \\ 0 & \text{if side} = 1 \end{cases}$

      Set $\hat{\underline{t}} = \begin{cases} \begin{bmatrix} 1 \\ 0 \end{bmatrix} & \text{if side} = 1,3 \\ \begin{bmatrix} 0 \\ 1 \end{bmatrix} & \text{if side} = 2,4 \end{cases}$

Loop over Element Boundaries

Call Shape Function $(e, \hat{\xi}_1^e, \hat{\xi}_2^e)$ which computes the NURBS basis functions R_a^e, their derivatives $\partial/\partial x\, R_a^e$ and $\partial/\partial y\, R_a^e$, the location $\vec{x}$, and the Jacobian matrix $\underline{\underline{J}}$ at the point $\vec{x}^e(\hat{\xi}_1^e, \hat{\xi}_2^e)$

Set $j_\Gamma = |\underline{\underline{J}}\, \underline{\hat{t}}|$

if Neumann (side, e) = 1

Neumann Boundary

Set $h_{loc} = h(\vec{x})$

for $a = 1, \ldots, n_{loc}$

Update $f_a^e = f_a^e + R_a^e * h_{loc} * \hat{w}_q * j_\Gamma$

endloop

elseif Robin (side, e) = 1

Robin Boundary

Set $B_{loc} = B(\vec{x})$

Set $u_{R,loc} = u_R(\vec{x})$

for $a = 1, \ldots, n_{loc}$

for $b = 1, \ldots, n_{loc}$

Update $k_{ab}^e = k_{ab}^e + B_{loc} * R_a^e * R_b^e * \hat{w}_q * j_\Gamma$

endloop

Update $f_a^e = f_a^e + B_{loc} * R_a^e * u_{R,loc} * \hat{w}_q * j_\Gamma$

endloop

endif

endif → endloop

endif

endloop

The only thing that remains to be specified is the function Shape Function which takes as input element number e and parent coordinates $\hat{\xi}_1$ and $\hat{\xi}_2$ and returns as output the NURBS basis functions R_a^e, their derivatives $\partial R_a^e/\partial x$ and $\partial R_a^e/\partial y$, the location $\vec{x}$, and the Jacobian matrix $\underline{\underline{J}}$ at the point $\vec{x}^e(\hat{\xi}_1^e, \hat{\xi}_2^e)$. Pseudocode for such a function is listed below.

Function Shape Function $(e, \tilde{\xi}_1, \tilde{\xi}_2)$
begin

Initialize $R_a^e = 0$, $\partial R_a^e/\partial x = 0$, $\partial R_a^e/\partial y = 0$ for $a = 1, \ldots, n_{loc}$

Initialize $\partial R_a^e/\partial \tilde{\xi}_1 = 0$, $\partial R_a^e/\partial \tilde{\xi}_2 = 0$ for $a = 1, \ldots, n_{loc}$

Initialize $\vec{x} = \vec{0}$, $\underline{\underline{J}} = \underline{\underline{0}}$, $w^b = 0$, $\partial w^b/\partial \tilde{\xi}_1 = 0$, $\partial w^b/\partial \tilde{\xi}_2 = 0$

Call Bernstein Basis And Derivs $(\tilde{\xi}_1, \tilde{\xi}_2)$ to obtain B_a^e, $\partial B_a^e/\partial \tilde{\xi}_1$, $\partial B_a^e/\partial \tilde{\xi}_2$

Pre-Compute

• $a = 1, \ldots, n_{loc}$

Weighting & Derivatives

for $a = 1, \ldots, n_{loc}$

Update $w^b = w^b + w_a^{b,e} * B_a^e$

Update $\partial w^b/\partial \tilde{\xi}_1 = \partial w^b/\partial \tilde{\xi}_1 + w_a^{b,e} * \partial B_a^e/\partial \tilde{\xi}_1$

Update $\partial w^b/\partial \tilde{\xi}_2 = \partial w^b/\partial \tilde{\xi}_2 + w_a^{b,e} * \partial B_a^e/\partial \tilde{\xi}_2$

endloop

Basis Functions & Parametric Derivatives

for $a = 1, \ldots, n_{loc}$

for $b = 1, \ldots, n_{loc}$

Update $R_a^e = R_a^e + w_a^e * C_{ab}^e * B_b^e / w^b$

Update $\frac{\partial R_a^e}{\partial \tilde{\xi}_1} = \frac{\partial R_a^e}{\partial \tilde{\xi}_1} + w_a^e * C_{ab}^e * \left(\frac{\partial B_b^e}{\partial \tilde{\xi}_1} / w^b - \frac{\partial w^b}{\partial \tilde{\xi}_1} * B_b^e / (w^b)^2 \right)$

Update $\frac{\partial R_a^e}{\partial \tilde{\xi}_2} = \frac{\partial R_a^e}{\partial \tilde{\xi}_2} + w_a^e * C_{ab}^e * \left(\frac{\partial B_b^e}{\partial \tilde{\xi}_2} / w^b - \frac{\partial w^b}{\partial \tilde{\xi}_2} * B_b^e / (w^b)^2 \right)$

endloop

endloop

Physical Space Quantities

for $a = 1, \ldots, n_{loc}$

Update $\vec{x} = \vec{x} + w_a^{b,e} * \vec{P}_a^{b,e} * B_a^e / w^b$

Update $J_{11} = J_{11} + (w_a^{b,e}) * (\vec{P}_a^{b,e})_1 * \left(\frac{\partial B_a^e}{\partial \tilde{\xi}_1} / w^b - \frac{\partial w^b}{\partial \tilde{\xi}_1} * B_a^e / (w^b)^2 \right)$

Update $J_{12} = J_{12} + (w_a^{b,e}) * (\vec{P}_a^{b,e})_1 * \left(\frac{\partial B_a^e}{\partial \tilde{\xi}_2} / w^b - \frac{\partial w^b}{\partial \tilde{\xi}_2} * B_a^e / (w^b)^2 \right)$

Update $J_{21} = J_{21} + (w_a^{b,e}) * (\vec{P}_a^{b,e})_2 * \left(\frac{\partial B_a^e}{\partial \xi_1^s} / w^b - \frac{\partial w^b}{\partial \xi_1^s} * B_a^e / (w^b)^2 \right)$

Update $J_{22} = J_{22} + (w_a^{b,e}) * (\vec{P}_a^{b,e})_2 * \left(\frac{\partial B_a^e}{\partial \xi_2^s} / w^b - \frac{\partial w^b}{\partial \xi_2^s} * B_a^e / (w^b)^2 \right)$

endloop

Set $\vec{\xi}^s_{,\vec{x}} = \begin{bmatrix} \frac{\partial \xi_1^s}{\partial x} & \frac{\partial \xi_1^s}{\partial y} \\ \frac{\partial \xi_2^s}{\partial x} & \frac{\partial \xi_2^s}{\partial y} \end{bmatrix} = \underline{\underline{J}}^{-1}$

Physical Derivatives

for $a = 1, \ldots, n_{loc}$

Set $\frac{\partial R_a^e}{\partial x} = \frac{\partial R_a^e}{\partial \xi_1^s} * \frac{\partial \xi_1^s}{\partial x} + \frac{\partial R_a^e}{\partial \xi_2^s} * \frac{\partial \xi_2^s}{\partial x}$

Set $\frac{\partial R_a^e}{\partial y} = \frac{\partial R_a^e}{\partial \xi_1^s} * \frac{\partial \xi_1^s}{\partial y} + \frac{\partial R_a^e}{\partial \xi_2^s} * \frac{\partial \xi_2^s}{\partial y}$

endloop

return $\{R_a^e\}_{a=1}^{n_{loc}}, \{\frac{\partial R_a^e}{\partial x}\}_{a=1}^{n_{loc}}, \{\frac{\partial R_a^e}{\partial y}\}_{a=1}^{n_{loc}}, \vec{x}, \underline{\underline{J}}$

end

It should be finally mentioned that the above function may be made much more efficient by exploiting the tensor product nature of NURBS. See:

M.J. Borden, M.A. Scott, J.A. Evans, and T.J.R. Hughes, "Isogeometric finite element data structures based on Bézier extraction of NURBS", IJNME, 87: 15-47, 2011

for more details.