Numerical Integration on Quadrilaterals

MECH 417, Rice University, J.E. Akin (revised 4/7/20)

[This section is like the one for integrating over curved triangles. The parametric space has just changed from a unit right triangle to a square in natural coordinates. Thus, just the tabulated integration data and the interpolation functions change.]

To evaluate the geometric properties of a part, like the mass moment of inertia matrix, just scalar polynomials must be integrated. For example, the mass moment of inertial about the y-axis is

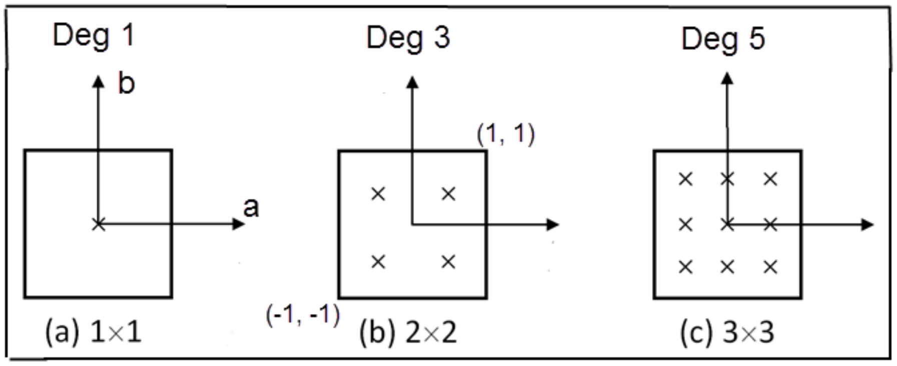
$$I_{yy} = \int_{\Omega} \rho\, x^2\, d\Omega$$

where $d\Omega$ is a physical differential region in 1-, 2-, or 3-D space, ρ is the corresponding mass density and x is the first spatial coordinate to a point in the curvilinear space, Ω. However, in FEA systems the integrand involves matrices along with some input scalar property.

Numerical integration replaces the integral with a sum where the integrand, $\rho\, x^2$, is evaluated at special tabulated points (quadrature points) and is multiplied by a special tabulated weight. The special tabulations are given in mathematical handbooks and/or online.

$$I_{yy} = e_{n_q} + \sum_{q=1}^{n_q} \rho_q\, x_q^2\, w_q$$

where $e_{n_q}$is the error resulting from the summation. If the integrand is a polynomial then the tabulated Gaussian quadrature rules are exact ($e_{n_q} \equiv 0$) when the total degree of the integrand is exactly integrated by 1-D rules in each of the two parametric directions. For line elements with polynomials the rule is $deg \leq (2n_1 - 1)$. However, for quadrilaterals (and brick elements) the rule is the product of the 1D edge rules along each of the two or three parametric directions, $n_q = n_1^2$ (or $n_q = n_1^3$).



Product of quadrature point rules

There are special rules that use fewer points for exact polynomial integrations than this product rule, but they are not covered here.

CAD and FEA systems use parametric geometry to model and simulate curvilinear parts. In those formulations the physical integral must be mapped to the corresponding integral in the parametric space, $\boxdot$. For example, the physical inertia integral changes to

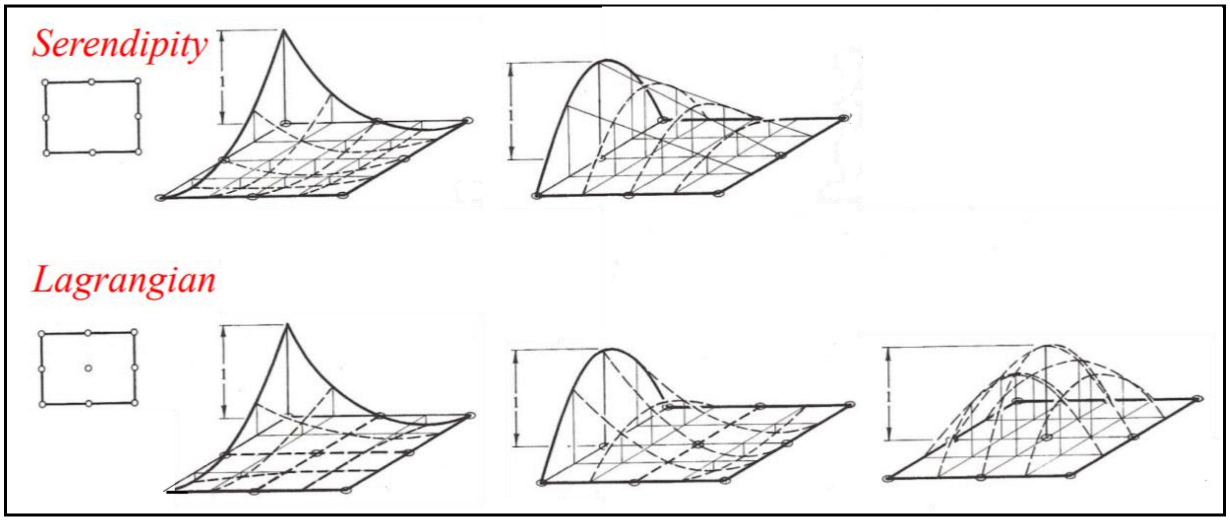
$$I_{yy} = \int_{\Omega} \rho\, x^2\, d\Omega = \int_{\boxdot} \rho(\boxdot)\, x^2(\boxdot)\, |J(\boxdot)| d\, \boxdot$$

where $|J(⊡)|$ denotes the determinant of the geometric Jacobian matrix of the mapping from physical space Ω to the parametric space ⊡. That Jacobian matrix generally varies over the parametric space and complicates the integrand, but that complication is more than offset by being able to automate the integration by using quadratures.

Quadrilateral and brick (hexahedral) elements usually use natural parametric coordinates which vary as $-1 \le a, b \le 1$ to define their interpolation functions. As shown in the next table the interpolation functions for the Serendipity family of quadrilateral elements, with equally spaced edge nodes, can be written in concise forms (where a_k denotes the non-dimensional parametric coordinate of node k, etc.).

Serendipity quadrilaterals in natural coordinates

Node Location		Interpolation Functions	Name
a_i	b_i	$H_i\,(a, b)$	
± 1	± 1	$(1 + aa_i)(1 + bb_i)/4$	Q4
± 1	± 1	$(1 + aa_i)(1 + bb_i)(aa_i + bb_i - 1)/4$	Q8
± 1	0	$(1 + aa_i)(1 - b^2)/2$	
0	± 1	$(1 + bb_i)(1 - a^2)/2$	
± 1	± 1	$(1 + aa_i)(1 + bb_i)\,[9(a^2 + b^2) - 10]/32$	Q12
± 1	$\pm 1/3$	$9(1 + aa_i)(1 - b^2)(1 + 9bb_i)/32$	
$\pm 1/3$	± 1	$9(1 + bb_i)(1 - a^2)(1 + 9aa_i)/32$	
± 1	± 1	$(1 + aa_i)(1 + bb_i)\,[4(a^2 - 1)\,aa_i + 4(b^2 - 1)\,bb_i + 3aba_i\,b_i\,]/12$	Q16
± 1	0	$2(1 + aa_i)(b^2 - 1)(b^2 - aa_i)/4$	
0	± 1	$2(1 + bb_i)(a^2 - 1)(a^2 - bb_i)/4$	
± 1	$\pm 1/2$	$4(1 + aa_i)(1 - b^2)(b^2 + bb_i)/3$	
$\pm 1/2$	± 1	$4(1 + bb_i)(1 - a^2)(a^2 + aa_i)/3$	
0	0	$(a^2 - 1)(b^2 - 1)$	



(Edge) Quadratic quadrilateral interpolation functions (see Appendix 2)

The original Gaussian quadrature data were tabulated in natural coordinates, and most of the literature uses this coordinate system. Note that in these coordinates the sum of the tabulated weights always equals four for quadrilaterals, which is the non-dimensional area of the parent square where each side is of length two (as shown above). The sum of the weights is eight for brick elements in natural coordinates.

Of course, these elements can also be interpolated in a unit coordinate system as the product of the two edge interpolations as sketched below:

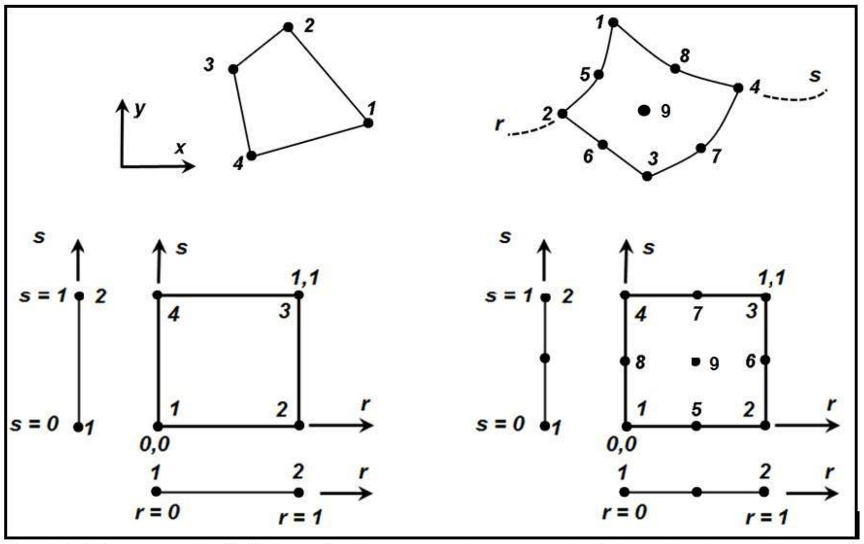


Figure 2.5-1 Four and eight node Lagrangian quadrilaterals in unit coordinates

Here, the interpolation functions for the Lagrange bi-linear four-node quadrilateral will also be developed, in unit coordinates, by using the product of the linear one-dimensional functions. From Fig. 2.5-1 the first two nodes on the quadrilateral have the interpolations at $s = 0$ multiplied times the two r-interpolations, and so on:

$$H_1(r,s) = H_1(r)H_1(s) = (1-r)(1-s) = 1 - r - s + rs$$
$$H_2(r,s) = H_2(r)H_1(s) = r(1-s) = r - rs$$
$$H_3(r,s) = H_2(r)H_2(s) = rs$$
$$H_4(r,s) = H_1(r)H_2(s) = (1-r)s = s - rs,$$

so

$$\boldsymbol{H}(r,s) = [(1-r-s+rs) \quad (r-rs) \quad (rs) \quad (s-rs)] \qquad (2.5\text{-}1)$$

These are identical to those derived by a different approach in Fig. 2.4-2, and they satisfy the requirement that $\sum_k H_k(r,s) \equiv 1$, as expected. In this coordinate system the measure of the non-dimensional space (and thus the sum of its numerical integration weights) is 1x1=1, not 4 required in natural coordinates.

As an example of numerical integration over a quadrilateral, assume that the physical region, Ω, is curved in the x-y space. Then, the area inertia integral is

$$\mathrm{I_{yy}} = \int_A \rho\, x^2\, dA = \int_{\boxdot} \rho(a,b)\, x^2(a,b)\, |J(a,b)|\, da\, db = \sum_{q=1}^{n_q} \rho(a_q,b_q)\ \ x^2(a_q,b_q)\, |J(a_q,b_q)|\, w_q$$

To illustrate this process consider just consider calculating the area of a curved quadrilateral in the x-y plane as shown below, with

$$A = \int_A dA = \int_A dx\, dy = \int_{-1}^{1}\int_{-1}^{1} |J(a,b)|\, da\, db\ = \sum_{q=1}^{n_q} |J(a_q,b_q)|\, w_q$$

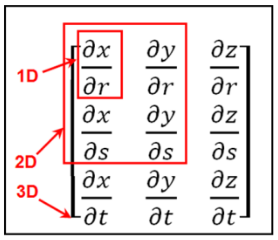
The eight x data points in 2-D space define an incomplete third degree polynomial (it is missing a^3 and b^3), as does the y-data. Those data can be interpolated over the curved quadrilateral by using a Q8 with eight nodes:

$$\mathrm{x(r,s)} = \sum_{\mathrm{k=1}}^{8} \mathrm{H_k(r,s)}\ \mathrm{x_k^e} = [\mathbf{H}(\mathrm{r,s})]\{\mathbf{x}\}^{\mathbf{e}}$$

and likewise for the y-data. Recall that the Jacobian matrix of the mapping from *x*,*y* to a, b is

$$[\boldsymbol{J}(a,b)] = \begin{bmatrix} \partial x/\partial a & \partial y/\partial a \\ \partial x/\partial b & \partial y/\partial b \end{bmatrix}$$

is just a sub-set of the 3D case, and an extension of the 1D case



and is almost always a variable in the parametric space. The determinant is

$$|\boldsymbol{J}^e(a,b)| = (\partial x/\partial a)(\partial y/\partial b) - (\partial x/\partial b)(\partial y/\partial a).$$

Those parametric derivatives are obtained from the above interpolations as

$$\frac{\partial \mathrm{x}(a,b)}{\partial a} = \sum_{\mathrm{k=1}}^{8} \frac{\partial \mathrm{H_k}(a,b)}{\partial a}\ \mathrm{x_k^e} = \left[\frac{\partial \mathbf{H}(a,b)}{\partial a}\right]\{\mathbf{x}\}^{\mathbf{e}}$$

This means that the Jacobian matrix at a point in an element can be found numerically as the product of the input coordinates matrix and the matrix of the parametric derivatives evaluated at the point:

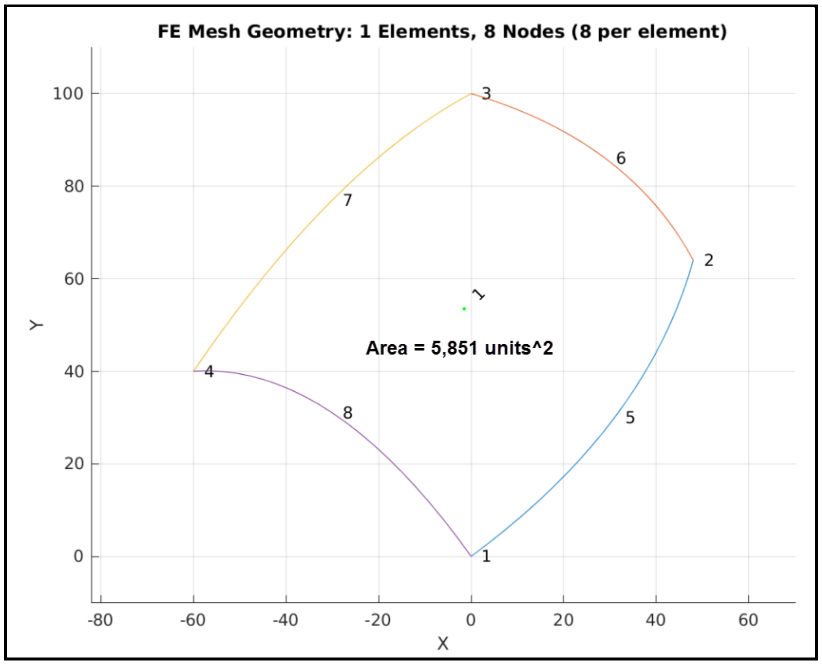
$$[\mathbf{J}(\mathrm{a_q,b_q})] = \begin{bmatrix} \partial x/\partial a & \partial y/\partial a \\ \partial x/\partial b & \partial y/\partial b \end{bmatrix}_q^e = \begin{bmatrix} \partial \boldsymbol{H}(\mathrm{a_q,b_q})/\partial r \\ \partial \boldsymbol{H}(\mathrm{a_q,b_q})/\partial s \end{bmatrix} \begin{bmatrix} x_1 & y_1 \\ x_2 & y_2 \\ x_3 & y_3 \end{bmatrix}^e$$

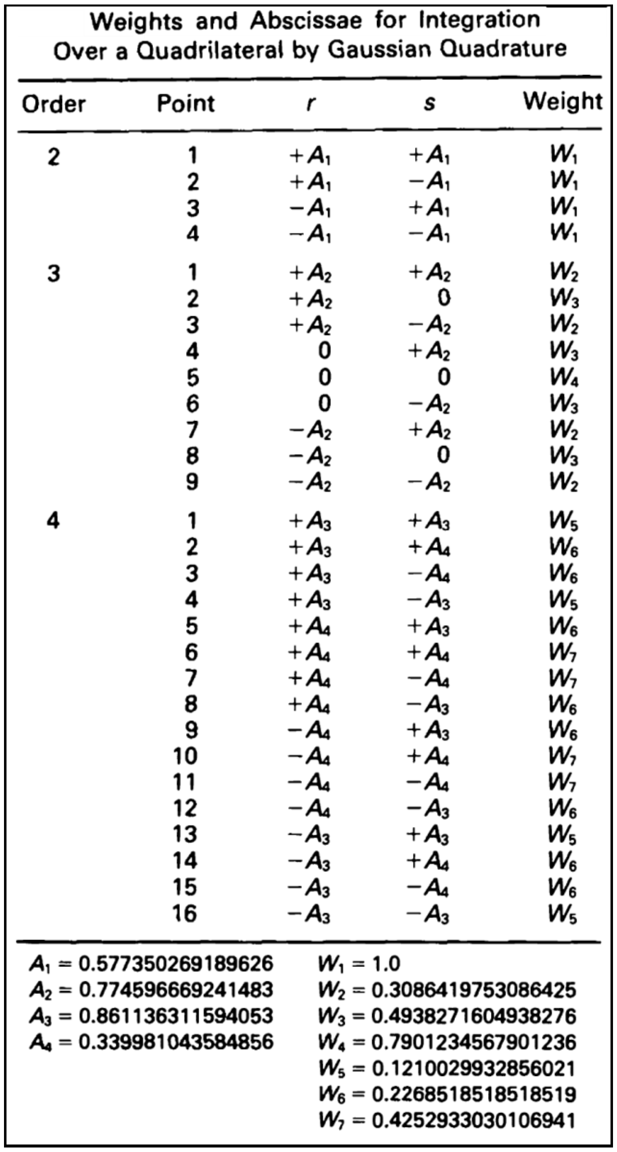
===

and numerically computing the determinant of that matrix the area becomes

$$\mathrm{A} = \sum_{\mathrm{q=1}}^{\mathrm{n_q}} |\mathbf{J}(\mathrm{a_q,b_q})|\ \mathrm{w_q}$$

The actual implementation is shown below (in three segments) in the download script *Area_Q4_or_Q8.m*.





Weights and Abscissae for Integration Over a Quadrilateral by Gaussian Quadrature

Order	Point	r	s	Weight
2	1	$+A_1$	$+A_1$	W_1
	2	$+A_1$	$-A_1$	W_1
	3	$-A_1$	$+A_1$	W_1
	4	$-A_1$	$-A_1$	W_1
3	1	$+A_2$	$+A_2$	W_2
	2	$+A_2$	0	W_3
	3	$+A_2$	$-A_2$	W_2
	4	0	$+A_2$	W_3
	5	0	0	W_4
	6	0	$-A_2$	W_3
	7	$-A_2$	$+A_2$	W_2
	8	$-A_2$	0	W_3
	9	$-A_2$	$-A_2$	W_2
4	1	$+A_3$	$+A_3$	W_5
	2	$+A_3$	$+A_4$	W_6
	3	$+A_3$	$-A_4$	W_6
	4	$+A_3$	$-A_3$	W_5
	5	$+A_4$	$+A_3$	W_6
	6	$+A_4$	$+A_4$	W_7
	7	$+A_4$	$-A_4$	W_7
	8	$+A_4$	$-A_3$	W_6
	9	$-A_4$	$+A_3$	W_6
	10	$-A_4$	$+A_4$	W_7
	11	$-A_4$	$-A_4$	W_7
	12	$-A_4$	$-A_3$	W_6
	13	$-A_3$	$+A_3$	W_5
	14	$-A_3$	$+A_4$	W_6
	15	$-A_3$	$-A_4$	W_6
	16	$-A_3$	$-A_3$	W_5

$A_1 = 0.577350269189626$	$W_1 = 1.0$
$A_2 = 0.774596669241483$	$W_2 = 0.308641975308 6425$
$A_3 = 0.861136311594053$	$W_3 = 0.493827160493 8276$
$A_4 = 0.339981043584856$	$W_4 = 0.790123456790 1236$
	$W_5 = 0.121002993285 6021$
	$W_6 = 0.226851851851 8519$
	$W_7 = 0.425293303010 6941$

```
function [Area] = Area_Q4_or_Q8 (n_n)   % Revised 3/19/20
%  Area of a Q4 or Q8 element by numerical integration
% using natural coordinates interpolation and quadrature

%  number of nodes per element n_n = 4 or 8 only
if (nargin == 0) ; n_n = 8 ; end ;  % if assign a default
n_e = 1           ;% number of elements in mesh (for plot)
n_m = n_n            ;% number of nodes in mesh (for plot)
n_t = 1     ;% number of element types in mesh (for plot)

% select which element model to use
if ( n_n == 4 )           ; % number of nodes per element
  nodes = [1  2    3    4]        ; % element connections
  x     = [0. 48.  0.  -60.]            ; % x-coordinates
  y     = [0. 64.  100. 40.]            ; % y-coordinates
  n_q = 1                      % number of quadrature pts
  % addpath /clear/www/htdocs/mech517/Akin_FEA_Lib
  % plot_input_2d_mesh (n_e, n_m, n_n, n_t, x, y, nodes);

elseif ( n_n == 8 )           ; % alternate choice of quad
  nodes = [1  2    3    4    5    6    7    8];% connections
  x     = [0. 48.  0. -60. 31. 29. -30. -30.]  ;% x-coord
  y     = [0. 64. 100. 40. 30. 86.  77.  31.]  ;% y-coord
  n_q = 4                                   % number of pts
  % q8_mesh_plot (x, y, nodes)         ; % plot curved el

else                               ; % invalid user input
  error ('Only valid function input is 4 or 8')
end              ; % if straight or curved element picked

% Set element coordinates as rectangular array
Coord = [x; y]'          ; % physical coordinates of nodes

% get the quadrature data over (-1,-1) to (1,1)
OK = [1, 4, 9, 16, 25]      ; % the only valid n_q inputs
if ( ~any (n_q == OK) )     ; % input not on required list
  error ('n_q invalid in qp_quad_nat_low')       ; % stop
end                                       ; % if bad data
```

1 of 3

```
switch n_q                     ; % branch on argument value
 case 1   ; % precision: exact for polynomial of degree 1
  a_q = [0] ; b_q = [0] ; w_q = [4] ;

 case 4   ; % precision: exact for polynomial of degree 3
  a_q = [-5.773502691896257e-1  5.773502691896257e-1 ...
         -5.773502691896257e-1  5.773502691896257e-1] ;
  b_q = [-5.773502691896257e-1 -5.773502691896257e-1 ...
          5.773502691896257e-1  5.773502691896257e-1] ;
  w_q = [ 1.000000000000000e+0  1.000000000000000e+0 ...
          1.000000000000000e+0  1.000000000000000e+0] ;

 case 9 ; % precision: exact for polynomial of degree 5
  a_q = [-7.745966692414834e-1  0.000000000000000e+0 ...
          7.745966692414834e-1 -7.745966692414834e-1 ...
          0.000000000000000e+0  7.745966692414834e-1 ...
         -7.745966692414834e-1  0.000000000000000e+0 ...
          7.745966692414834e-1] ;
  b_q = [-7.745966692414834e-1 -7.745966692414834e-1 ...
         -7.745966692414834e-1  0.000000000000000e+0 ...
          0.000000000000000e+0  0.000000000000000e+0 ...
          7.745966692414834e-1  7.745966692414834e-1 ...
          7.745966692414834e-1] ;
  w_q =  [3.086419753086420e-1  4.938271604938271e-1  ...
          3.086419753086420e-1  4.938271604938271e-1  ...
          7.901234567901234e-1  4.938271604938271e-1  ...
          3.086419753086420e-1  4.938271604938271e-1  ...
          3.086419753086420e-1];

 otherwise     ; % use function that generated above list
  [a_q, b_q, w_q] = qp_rule_nat_quad (n_q); % use 1D data
end                           ; % switch on number of points
```

2 of 3

```
Area = 0.0                       ; % initialiaze area sum
for q = 1: n_q ; % loop over integration points ---> --->
  a = a_q(q) ; b = b_q(q) ; w = w_q(q)   ; % get pts & wt
  % H   = element interpolation functions at a,b
  % DLH = interpolation local derivatives at a,b
  switch n_n % select from available library of elements
   case {4} % four node quadrilateral, -1 <= a,b <= 1, Q4
   % type interpolation functions, 1 x 4, 1-2-3-4 CCW
    H = [(1-a)*(1-b), (1+a)*(1-b), ...
         (1+a)*(1+b), (1-a)*(1+b)]/4 ;
  %               element type parametric derivatives, 2 x 4
    DLH = [-(1-b),  (1-b), (1+b), -(1+b); ...   % dH/da
           -(1-a), -(1+a), (1+a),  (1-a)]/4  ;  % dH/db

   case {8}  % eight node quadrilateral, 1 x 8,        b
  % (See qp_rule_nat_quad for tabulated data)     % 4 7 3
  % corners 1-2-3-4 CCW, mid 5-6-7-8.             % 8   6 a
  % for-1 <= a,b <= 1                             % 1 5 2
    a_p = 1 + a; a_m = 1 - a; b_p = 1 + b; b_m = 1 - b;
     H = [a_m*b_m*(a_m+b_m-3), a_p*b_m*(a_p+b_m-3), ...
          a_p*b_p*(a_p+b_p-3), a_m*b_p*(a_m+b_p-3), ...
          2*b_m*(1-a*a), 2*a_p*(1-b*b), ...
          2*b_p*(1-a*a), 2*a_m*(1-b*b)]/4 ; % H(a,b)
  %               element type parametric derivatives, 2 x 8
     DLH = [-b_m*(a_m+a_m+b_m-3),  b_m*(a_p+a_p+b_m-3), ...
          b_p*(a_p+a_p+b_p-3), -b_p*(a_m+a_m+b_p-3), ...
         -4*a*b_m, 2*(1-b*b), -4*a*b_p, -2*(1-b*b);   % dH/da
         -a_m*(b_m+a_m+b_m-3), -a_p*(b_m+a_p+b_m-3), ...
          a_p*(b_p+a_p+b_p-3),  a_m*(b_p+a_m+b_p-3), ...
         -2*(1-a*a), -4*b*a_p, 2*(1-a*a), -4*b*a_m]/4;% dH/db
  end ; % switch on number of nodes

% calculate the Jacobian, for this point only
  Jacobian = DLH * Coord  ; % 2 by 2 Jacobian matrix at q
  Det_J    = det (Jacobian)   ; % scalar determinate at q
  Area = Area + Det_J * w   ;
end ; % loop over integration points  <--- <--- <--- <---
fprintf('Area of quadrilateral is %9.3e \n', Area)% result
% end Area_Q4_or_Q8 ========================================
```

3 of 3

Original question:

Note that in the quadrature point loop (segment 3) that the values of the interpolation functions, $\boldsymbol{H}(a_q, b_q)$, were also evaluated but not used. In finite element applications they are almost always needed in formulating element matrices and/or in post-processing the solution. It is usually more efficient to store items evaluated during computing the element matrices rather than re-compute them in post-processing the solution. In post-processing it is usually important to know the physical (x, y) location of each quadrature point. Using the element nodal coordinates a matrix dot produce gives those data

$$x(a_q, b_q) = \sum_{k=1}^{8} H_k(a_q, b_q)\ x_k^e = [\mathbf{H}(a_q, b_q)]\{\mathbf{x}\}^e$$

$$y(a_q, b_q) = \sum_{k=1}^{8} H_k(a_q, b_q)\ y_k^e = [\mathbf{H}(a_q, b_q)]\{\mathbf{y}\}^e$$

The original question was how to calculate the mass inertia term

$$I_{yy} = \sum_{q=1}^{n_q} \rho_q\, x_q^2\ w_q$$

Assuming the mass per unit area is constant, say rho, the physical location would be obtained with

x_q = **H * Coord** ; % x physical point

and the increment to the inertia term would be

I_yy = I_yy + rho * x_q^2 * Det_J * w ; % increment inertia

Of course, that calculation would increase the highest degree to be integrated and thereby increase the minimum value of the number of quadrature points required as an input argument.

Appendix 1, Examples:

Example 4.2-6 Given: A quadrilateral area in physical space is defined in 'natural coordinate' parametric space by the mapping $x(a, b) = (-3 + 27\,a - 27\,b + 3\,a\,b)$
and $y\,(a, b) = (51 + 31\,a + 19\,b - a\,b)$ where $-1 \le a, b \le +1$.

Verify that the variable Jacobian matrix of the mapping is

$$J(a, b) = \begin{bmatrix} (27 + 3\,b) & (31 - b) \\ (-27 + 3\,a) & (19 - a) \end{bmatrix},$$

and determine its determinant.

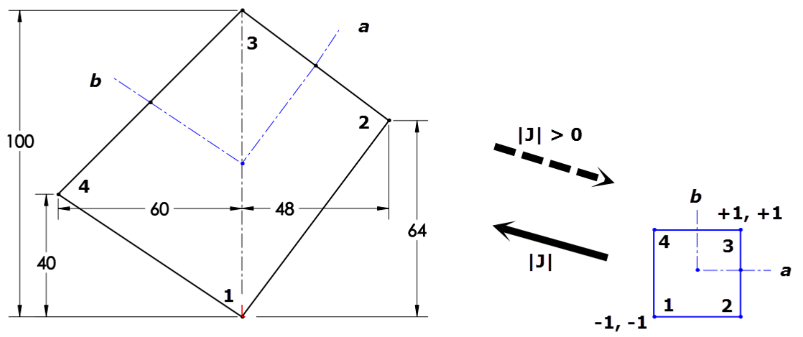
Solution: Calculating the partial derivatives with respect to a: $\partial x/\partial a = (0 + 27 - 0 + 3\,b)$ and $\partial y/\partial a = (0 + 31 + 0 - b)$. These are the terms in the first row of the Jacobian matrix. Repeating the process for derivatives with respect to b gives the cited matrix:

$$\boldsymbol{J}(a, b) = \begin{bmatrix} \partial x/\partial a & \partial y/\partial a \\ \partial x/\partial b & \partial y/\partial b \end{bmatrix} = \begin{bmatrix} (27 + 3\,b) & (31 - b) \\ (-27 + 3\,a) & (19 - a) \end{bmatrix}.$$

Thus, $|J(a, b)| = (27 + 3\,b)(19 - a) - (31 - b)(-27 + 3\,a) = 1{,}350 - 120\,a + 30\,b$. The determinant is not constant, but it is positive for the given range of *(a, b)* values. Since the determinant is positive, $|J(a, b)| > 0$, this mapping is called invertible. Valid finite element shapes should be invertible.

Example 4.2-7 Given: The natural coordinate square $-1 \le a, b \le +1$ is mapped into a physical quadrilateral by $x(a, b) = (-3 + 27\,a - 27\,b + 3\,a\,b)$ and $y\,(a, b) = (51 + 31\,a + 19\,b - a\,b)$, in meters. Evaluate the mapping at the four parametric corners to find the physical coordinates of the corners of the quadrilateral. Use the mapping to determine the physical area of the quadrilateral.

Solution: Directly substituting a variable b at $a = \pm 1$ gives two sides connecting the physical coordinates of the vertices. Substituting a variable a at $b = \pm 1$ completes the figure below:



a	b	x (a, b)	y (a, b)
-1	-1	0	0
-1	+1	48	64
+1	+1	0	100
+1	-1	-60	40

The area is given by the integral $A = \int_A dA = \int_{-1}^{1}\int_{-1}^{1} |J(a,b)|\, da\, db$. The determinant for this area, with the units m^2, was developed in Ex. 4.2-4. Substituting gives

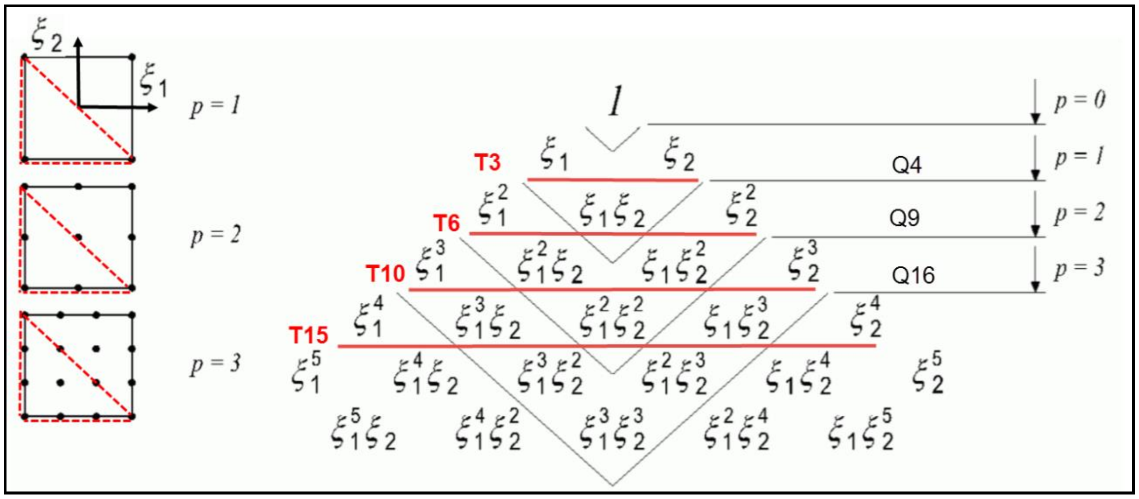
$$A = \int_A dA = \int_{-1}^{1}\int_{-1}^{1} |J(a,b)|\, da\, db = m^2 \int_{-1}^{1}\int_{-1}^{1} (1{,}350 - 120\, a + 30\, b)\, da\, db = 5{,}400\, m^2$$

Appendix 2, 2D Interpolation Terms

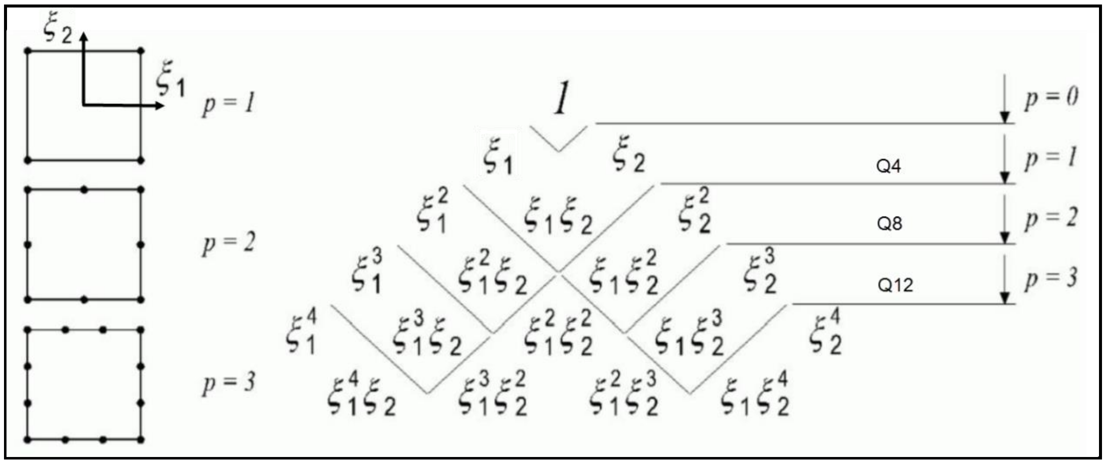
Triangular elements always use ‘complete polynomials’ (defined below) whose degree is defined by the number of nodes on an edge. For example, a quadratic T6 element has three nodes on each edge and thus is a complete 1D polynomial of degree 2 on the edge. Its interior interpolation contains the addition product(s) needed to define a complete 2D polynomial (the red lines below); and nothing of higher degree.

Quadrilateral elements likewise have complete 1D polynomials on each edge. They are therefore compatible with a triangular element of the same 1D edge degree, even if they are curved. Quadrilateral elements always have interior interpolations that are of a higher 2D degree than their 1D edge degree, and they are always incomplete polynomials. There are two types of quadrilaterals, the original Lagrangian quads that always have interior nodes and the later Serendipity quads that avoid most interior nodes (due to a lack of computer power in 1960’s).

Both types have higher degree polynomial terms on their interior than triangular elements with the same number of edge nodes. Therefore, quadrilaterals give more accurate solutions than triangles. However, it is very much easier to automatically create triangular meshes than quadrilateral meshes. Thus, use a large number of triangles.

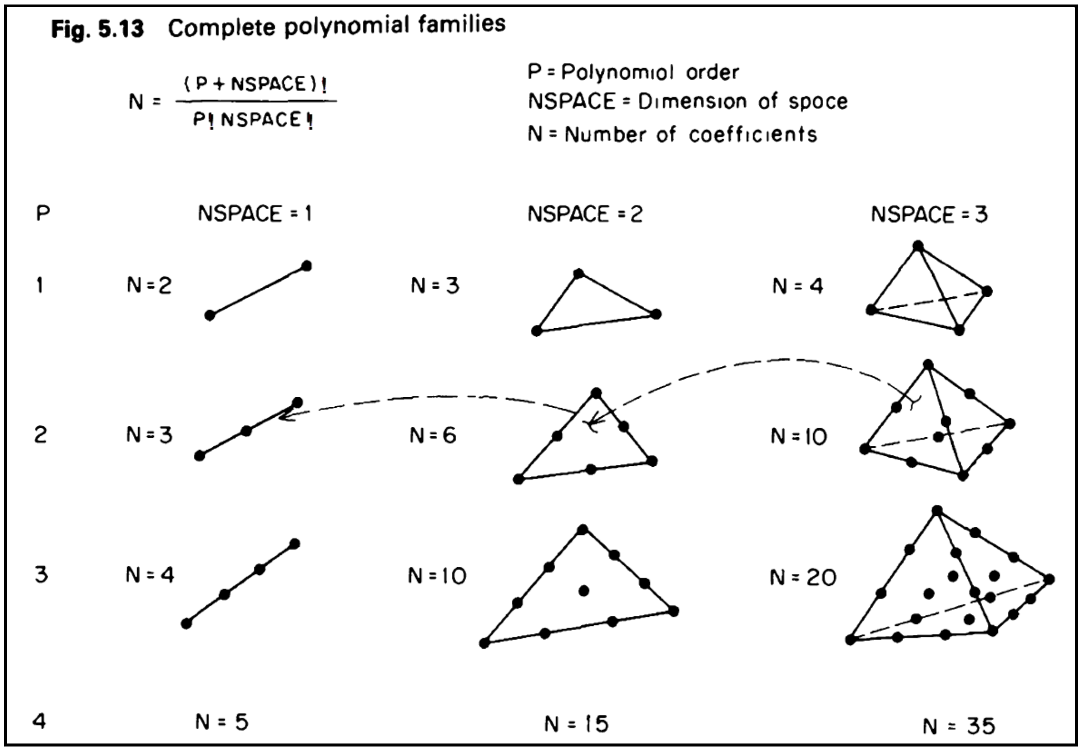

ξ_2
ξ_1
$p = 1$
$p = 2$
$p = 3$
1
$p = 0$
ξ_1 ξ_2
T3
Q4
ξ_1^2 $\xi_1\xi_2$ ξ_2^2
T6
Q9
ξ_1^3 $\xi_1^2\xi_2$ $\xi_1\xi_2^2$ ξ_2^3
T10
Q16
ξ_1^4 $\xi_1^3\xi_2$ $\xi_1^2\xi_2^2$ $\xi_1\xi_2^3$ ξ_2^4
T15
ξ_1^5 $\xi_1^4\xi_2$ $\xi_1^3\xi_2^2$ $\xi_1^2\xi_2^3$ $\xi_1\xi_2^4$ ξ_2^5
$\xi_1^5\xi_2$ $\xi_1^4\xi_2^2$ $\xi_1^3\xi_2^3$ $\xi_1^2\xi_2^4$ $\xi_1\xi_2^5$

Lagrange quadrilateral interpolation polynomial terms


ξ_2
ξ_1
$p = 1$
$p = 2$
$p = 3$
1
$p = 0$
ξ_1 ξ_2
Q4
ξ_1^2 $\xi_1\xi_2$ ξ_2^2
Q8
ξ_1^3 $\xi_1^2\xi_2$ $\xi_1\xi_2^2$ ξ_2^3
Q12
ξ_1^4 $\xi_1^3\xi_2$ $\xi_1^2\xi_2^2$ $\xi_1\xi_2^3$ ξ_2^4
$\xi_1^4\xi_2$ $\xi_1^3\xi_2^2$ $\xi_1^2\xi_2^3$ $\xi_1\xi_2^4$

Serendipity quadrilateral interpolation polynomial terms

Fig. 5.13 Complete polynomial families



Appendix 3: Convert Gauss' 1D rule to 2D square rule

```
function [r_q, s_q, w_q] = qp_rule_nat_quad (n_q) % ===============
%     tables of quadrature point locations and weights for
%  quadrilaterals interpolated in natural coords, -1 <= r,s <= 1

% n_1 = number of 1-D points in each direction
% n_q = 2-D number of quadrature points required, = n_1^2
% r_q = all of first parametric quadrature coordinates
% s_q = all of second parametric quadrature coordinates
% w_q = weight at all quadrature points in n_p

r_q = zeros (n_q, 1); s_q = zeros (n_q, 1) ; w_q = zeros (n_q, 1) ;

if ( n_q == 1 )          ; % exact for constant or linear polynomial
  r_q = [ 0 ] ; s_q = [ 0 ] ; w_q = [ 4 ] ;   % centroid point data

% tensor products of 1-D rule
elseif ( n_q == 4 | n_q == 9 | n_q == 16 | n_q == 25 ); % tabulated
  n_1 = fix ( sqrt ( n_q ) )                   ; % size of 1-D rule
  [r_1, w_1] = qp_rule_Gauss (n_1)            ; % get 1-D rule data
  k = 0                            ; % initialize quad point number
  for i = 1 : n_1               ; % loop over points in s-direction
    for j = 1 : n_1             ; % loop over points in r-direction
      k = k + 1               ; % point number in the quadrilateral
      w_q (k) = w_1 (i) * w_1 (j)        ; % product of 1-D weights
      r_q (k) = r_1 (j)           ; % r-coordinate in quadrilateral
      s_q (k) = r_1 (i)           ; % s-coordinate in quadrilateral
    end                            ; % for j point in quadrilateral
  end                              ; % for i point in quadrilateral
else                          ; % update the tables or elseif above
  error ('\nERROR quad rule not tabulated for these points')
end                             ; % if number of quadrature points
% end qp_rule_nat_quad % ==========================================
```