Universitat Politècnica de Catalunya
Numerical Methods in Engineering
Computational Solid Mechanics and Dynamics

Convergence requirements

Assignment 5

Eduard Gómez
March 13, 2020

Contents

1 Assignment 4.1

1.1 Statement

The isoparametric definition of the straight–node bar element in its local system $\underline{x}$ is:

$$\begin{bmatrix} 1 \\ \bar{x} \\ \bar{u} \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 \\ \bar{x}_1 & \bar{x}_2 & \bar{x}_3 \\ \bar{u}_1 & \bar{u}_2 & \bar{u}_3 \end{bmatrix} \begin{bmatrix} N_1^e(\xi) \\ N_2^e(\xi) \\ N_2^e(\xi) \end{bmatrix} \tag{1}$$

Here ξ is the isoparametric coordinate that takes the values –1, 1 and 0 at nodes 1, 2 and 3 respectively, while N_1^e, N_2^e and N_3^e are the shape functions for a bar element.

For simplicity, take $\bar{x}_1 = 0$, $\bar{x}_2 = L$ and $\bar{x}_3 = \frac{1}{2}L \pm \alpha L$. Here L is the bar length and α a parameter that characterizes how far node 3 is away from the midpoint location $\bar{x} = \frac{1}{2}L$.

Show that the minimum α (minimal in absolute value sense) for which $J = d\bar{x}/d\xi$ vanishes at a point in the element are ±¼ (the quarter points). Interpret this result as a singularity by showing that the axial strain becomes infinite at an end point

1.2 Solution

Let's start off by defining the shape functions in ispoarametric space:

$$\left.\begin{aligned} N_1(\xi) &= \frac{1}{2}\xi(1-\xi) \\ N_2(\xi) &= \frac{1}{2}\xi(1+\xi) \\ N_3(\xi) &= 1-\xi^2 \end{aligned}\right\} \tag{2}$$

Let's now opearate the second row in equation 1:

$$\bar{x} = \bar{x}_1 N_1(\xi) + \bar{x}_2 N_2(\xi) + \bar{x}_3 N_3(\xi) \tag{3}$$

$$= \frac{1}{2}L\xi(1+\xi) + \left(\frac{1}{2}L + \alpha L\right)(1-\xi^2) \tag{4}$$

$$= \frac{1}{2}L\left(1 + 2\alpha - 2\alpha\xi^2\right) \tag{5}$$

As expected, it looks like a parabola for all cases where the points are unequally spaced (i.e. $\alpha \neq \frac{1}{2}$). Let's now compute the jacobian:

$$J(\xi) = \frac{d\bar{x}}{d\xi} = \frac{1}{2}L\left(1 - 4\alpha\xi\right) \tag{6}$$

Once again, the mapping is affine for equally spaced points. Let's now find what α causes the Jacobian to vanish at some point of ξ. The jacobian is linear so for the critical value of α either end of the domain

$\xi \in [-1, 1]$ will be zero. Hence:

$$J(\pm 1) = 0 \tag{7}$$
$$\frac{1}{2} L \left(1 \pm 4\alpha_0\right) = 0 \tag{8}$$
$$1 \pm 4\alpha_0 = 0 \tag{9}$$
$$\alpha_0 = \mp \frac{1}{4} \tag{10}$$

Therefore our critical value is $|\alpha| = \frac{1}{4}$. Let's see how this affects the strain. We have that:

$$\varepsilon(x) = \frac{du}{dx} \tag{11}$$
$$= \sum_{i=1}^{3} \frac{dN_i}{dx} u(x_i) \tag{12}$$
$$= \sum_{i=1}^{3} \frac{d\xi}{dx} \frac{dN_i}{d\xi} u(x_i) \tag{13}$$
$$= J^{-1}(\xi(x)) \sum_{i=1}^{3} \frac{dN_i}{d\xi} u(x_i) \tag{14}$$

When the Jacobian vanishes, its reciprocal becomes unbounded, therefore so does the strain.

2 Assignment 4.2

2.1 Statement

Extend the results obtained from the previous Exercise for a 9-node plane stress element. The element is initially a perfect square, nodes 5,6,7,8 are at the midpoint of the sides 1–2, 2–3, 3–4 and 4–1, respectively, and 9 at the center of the square.

Move node 5 tangentially towards 2 until the Jacobian determinant at 2 vanishes. This result is important in the construction of "singular elements" for fracture mechanics.

2.2 Solution

This section can quickly become tedious due to long algebraic manipulation, hence I wrote a Matlab script to solve it for me. Partial results are shown nevertheless. We'll define the mapping as $M : \boldsymbol{\xi} \mapsto \boldsymbol{x}$. We start with a unit square in natural space $[0,0] \times [1,1]$ and obtain it's mapping to isoparametric space using the same method as before:

$$M(\xi,\eta) = \sum_{i=1}^{9} \begin{bmatrix} x_i \\ y_i \end{bmatrix} N(\xi,\eta) \tag{15}$$

where x_i and y_i are the positions of the nodes in natural coordinates. Applying the previous equation returns:

$$M(\xi,\eta) = \frac{1}{2} \begin{bmatrix} (\xi+1)\left(\eta^2\alpha - \eta\alpha + \xi\eta\alpha - \xi\eta^2\alpha + 1\right) \\ \eta + 1 \end{bmatrix} \tag{16}$$

Once again α is the source of non-linearity. Thaking the gradient in iso-parametric space yields:

$$J(\xi,\eta) = \nabla^{\text{iso}} M(\xi,\eta) = \frac{1}{2} \begin{bmatrix} 2\xi\eta\alpha - 2\xi\eta^2\alpha + 1 & -\alpha\left(2\eta - 1\right)\left(\xi^2 - 1\right) \\ 0 & 1 \end{bmatrix} \tag{17}$$

Today we're interested in the determinant of the jacobian:

$$|J(\xi,\eta)| = 2\xi\eta\alpha - 2\xi\eta^2\alpha + 1 \tag{18}$$

We particularly want the jacobian to vanish at node 2, hence $\boldsymbol{\xi} = [1,-1]^T$:

$$|J(1,-1)| = \frac{1}{4} - \alpha \tag{19}$$

We see that it vanishes at $\alpha = \frac{1}{4}$.

A Appendix

A.1 Matlab program for assignment 5.2

The program looks like the following. The functions used to turn the expressions into LaTeX were developed by me as well, so all the work is original. Function `shape_fun_quad_9` is also mine and shown in appendix A.3

```
%%%%%%%%%%%% Using functions to help write the report: %%%%%%%%%%%%
addpath('MatlabLaTeX');
addpath('MatlabLaTeX/format_sym_expression')

%%%%%%%%%%%%%%%%%%%%%%%%% Variable declaration %%%%%%%%%%%%%%%%%%%%%
% Using this strange name convention so format_sym_expression() can
% turn them into latex variables \xi, \eta and \alpha
xi = sym('__BS__xi','real');
eta = sym('__BS__eta','real');
alpha = sym('__BS__alpha','positive');
xi_critical  =  1;
eta_critical = -1;

%%%%%%%%%%%%%%%%%%%%%%%% Square to be studied: %%%%%%%%%%%%%%%%%%%%%
%     1  2  3  4     5     6  7  8  9
X = [ 0  1  1  0 .5+alpha  1 .5  0 .5;
      0  0  1  1     0    .5  1 .5 .5];

%%%%%%%%%%% Obtaining natural to isoparametric mapping:  %%%%%%%%%%
map = 0;
for shape_fun = 1:9
    map = map + X(:,shape_fun)*shape_fun_quad_9(shape_fun, xi, eta);
end
map = simplify(map);
matrixLaTeX('map.tex',map,'%s');
disp(' ')
disp('# Map stored in ');
disp('map.tex');

%%%%%%%%%%%%%%%%%%%% Obtaining jacobian matrix: %%%%%%%%%%%%%%%%%%%
Jmat = 0*sym('J');
isop = {xi, eta};

for i = 1:2
    for j = 1:2
        Jmat(i,j) = simplify(diff(map(i), isop{j}));
    end
end
matrixLaTeX('jacobian.tex',Jmat,'%s');
disp(' ')
disp('# Jacobian matrix stored in');
disp('jacobian.tex');
```

```

%%%%%%%%%%%%%%%%%%%%%%% Obtaining jacobian: %%%%%%%%%%%%%%%%%%%%%%
disp(' ')
disp('# |J(xi,eta)|:')
J = simplify(det(Jmat));
disp(format_sym_expression(J));

%%%%%%%%%%%%%%%%%%%% Evaluating at point 2 %%%%%%%%%%%%%%%%%%%%
Jcritical = subs(subs(J,xi,xi_critical),eta,eta_critical);
disp(' ')
disp('# |J(xi,eta)| @  critical node:')
disp(format_sym_expression(Jcritical));

%%%%%%%%%%%%%%%% Solving for Jcritical = 0 %%%%%%%%%%%%%%%%%%%%%
alpha_critical = solve(Jcritical == 0, alpha);
disp(' ')
disp('# Critical alpha:')
disp(format_sym_expression(alpha_critical));
```

A.2 Program outputs

The output of the program looks like such:

```
# Map stored in
map.tex

# Jacobian matrix stored in
jacobian.tex

# |J(xi,eta)|:
\frac{\xi\eta\alpha}{2}-\frac{\left(\xi\eta^2\alpha\right)}{2}+\frac{1}{4}

# |J(xi,eta)| @  critical node:
\frac{1}{4}-\alpha

# Critical alpha:
\frac{1}{4}
```

Where the two `tex` files are the matrices in equations 16 and 17.

A.3 Shape functions subroutine

```
function z = shape_fun_quad_9(i,X,Y)
    % Program to calculate shape functions on a plane quadrilateral with
    % nine nodes in isoparametric space.
    % INPUTS
    % - i is the shape function N_i to evaluate. Only one value.
    % - X is a an array, vector or variable to evauate on
    % - Y is a an array, vector or variable to evauate on
    % OUTPUTS
    % - z a an array, vector or variable of shape function z = N_i(i,X,Y);
    if(size(i,1) ~= 1 || size(i,2) ~= 1)
        error('i must be a single number, not a vector or array');
    elseif(i < 1 || i > 9)
        error('i must be within 1 and 9');
    end
    if(size(X,1) ~= size(Y,1) || size(X,2) ~= size(Y,2))
        error ('X and Y must be the same size');
    end

    %            1  2  3  4  5  6  7  8  9
    X_nodes = [-1  1  1 -1  0  1  0 -1  0];
    Y_nodes = [-1 -1  1  1 -1  0  1  0  0];
    for a = size(X,1):-1:1
        for b = size(X,2):-1:1
            x = X(a,b);
            y = Y(a,b);

            x_node = X_nodes(i);
            y_node = Y_nodes(i);

            vals = [-1,0,1];

            X0 = vals(vals~=x_node);
            Y0 = vals(vals~=y_node);

            if i < 5
                z0 = 0.25;
            elseif i<9
                z0 = -0.5;
            else
                z0 = 1;
            end
            z(a,b) = z0 * (X0(1) - x)*(X0(2) - x)*(Y0(1) - y)*(Y0(2) - y);
        end
    end
end
```