MKM321 2D FEM USER GUIDE

Daniel N. Wilke

A finite element code consists of the following three parts

1. Pre-processor that defines a problem by constructing an input file
2. Finite element solver that takes an input file and solves the problem
3. Finite element solver returns result for interpretation

The 2D FEM code supplied by the module:
fem_main_program.py
which requires three additional modules
preprocess.py
elements.py
postprocess.py

Inside **fem_main_program.py, launch_fem** can be loaded by typing:

from fem_main_program import launch_fem

The main function in *launch_fem* is FEM2D that requires the name of the input file as input and optionally the keyword MagFac to scale the displacements for visualization purposes e.g.
U, ReactionForces, VonMises, SXX, SYY, SXY = launch_fem('q8_patch_translate',MagFac=1, IntegrationType = 'FI')

The function solves the problem and then returns the displacement vector U, ReactionForces, Von Mises stress, Stress along x (SXX), Stress along Y (SYY) and Shear stress (SXY) as output. In addition post-processing such as plotting of the displacement and stress fields of the computed result is done. MagFac is a factor by which the displacement is multiplied. IntegrationType indicates full integration (FI) or reduced integration (RI).

Input File Structure
The input file has the following structure
***NODE**
NodeNumber Xcoordinate Ycoordinate
***ELTYPE**
Even integer - plane stress
Odd integer - plane strain
***ELEMENT**
ElementNumber Node1 Node2 Node3 Node4
(Four Node Numbers Q4 element must follow an anti-clockwise numbering)
ElementNumber Node1 Node2 Node3 Node4 Node5 Node6 Node7 Node8
Eight Node Numbers Q8 element must follow an anti-clockwise numbering)
In addition:
Node5 between Node1 and Node2
Node6 between Node2 and Node3
Node7 between Node3 and Node4
Node8 between Node4 and Node1
***BOUNDARY**
Specify prescribed displacement boundary conditions
NodeNumber Direction(1 for x and 2 for y) PrescribedDisplacementValue(0 implies fixed in specified direction)
***EMODULUS**
Specify the Young’s modulus for the material
***POISSON**
Specify Poisson’s ratio for the material
***THICKNESS**

Specify the element thickness, this code only supports a constant thickness over the whole structure.

***CLOAD**
NodeNumber Direction(1 for x and 2 for y) PrescribedForce(Positive along positive axis direction, and negative along negative axis direction)
***END**
(Optional) Ends Input File

EXAMPLE INPUT FILE ONE Q8 ELEMENT

```
*NODE
1 0.0 -0.5
2 2.0 -0.5
3 2.0 0.5
4 0.0 0.5
5 1.0 -0.5
6 2.0 0.0
7 1.0 0.5
8 0.0 0.0
*ELTYPE
 5
*ELEMENT
1 1 2 3 4 5 6 7 8
*BOUNDARY
1 1 0
1 2 0
4 1 0
8 1 0
*EMODULUS
210000.0
*POISSON
0.3
*THICKNESS
1.0
*CLOAD
2 1 10
6 1 40
3 1 10
```

Input file contains eight nodes to construct one 8-noded elements.
The problem is plane strain.
Node 1, 4 and 8 are constrained in x
Node 1 is also constrained in y

Units are implied by the user. For example, if mm are chosen for position of the nodes and N for the forces specified in CLOAD then stresses are computed as MPa. The user must then ensure that thickness is in mm and EMODULUS in MPa. Reminder Poisson’s ration is unitless.

A concentrated load of 10 N is applied at nodes 2, 6 and 3 in the positive x-direction.

EXAMPLE OUTPUT RESULTS ONE Q8 ELEMENT

The following vectors are returned. Firstly the displacement U:

```
array([[ 0.00000000e+00],
       [ 0.00000000e+00],
       [ 5.20000000e-04],
       [ 5.89534931e-19],
       [ 5.20000000e-04],
       [-1.11428571e-04],
       [ 0.00000000e+00],
       [-1.11428571e-04],
       [ 2.60000000e-04],
       [ 3.79470760e-19],
       [ 5.20000000e-04],
       [-5.57142857e-05],
       [ 2.60000000e-04],
       [-1.11428571e-04],
       [ 0.00000000e+00],
       [-5.57142857e-05]])
```

Reaction Forces:

```
matrix([[-1.00000000e+01],
        [ 7.10542736e-15],
        [-1.00000000e+01],
        [-4.00000000e+01]])
```

VonMises:

```
array([[53.3291665, 53.3291665, 53.3291665, 53.3291665]])
```

SXX:

```
matrix([[60., 60., 60., 60.]])
```

SYY:

```
matrix([[-4.88794081e-14,  3.84763404e-14,  3.11158397e-14,
          7.70893742e-15]])
```

SXY:

```
matrix([[ 4.28965721e-14,  1.25024305e-14, -1.63518627e-14,
          2.54869343e-15]])
```

Displacement

The x-displacement for node number A is given by index number: **2A - 2** (2 times A minus 2) i.e. for A = 1 the x-displacement is given by index 0 which is 0.00000000e+00. Recall node 1 is constrained to move in x and y.

The y-displacement for node number A is given by index number: **2A - 1** (2 times A minus 1) i.e. for A = 1 the y-displacement is given by index number 1 which is 0.00000000e+00. Recall node 1 is constrained to move in x and y, we can see that both are satisfied exactly.

Similarly A = 5 for x-displacement is index 8 which gives `2.60000000e-04`.

Similarly A = 5 for y-displacement is index 9 which gives `3.79470760e-19`.

```
array([[ 0.00000000e+00],
       [ 0.00000000e+00],
       [ 5.20000000e-04],
       [ 5.89534931e-19],
       [ 5.20000000e-04],
       [-1.11428571e-04],
       [ 0.00000000e+00],
       [-1.11428571e-04],
       [ 2.60000000e-04],
       [ 3.79470760e-19],
       [ 5.20000000e-04],
       [-5.57142857e-05],
       [ 2.60000000e-04],
       [-1.11428571e-04],
       [ 0.00000000e+00],
       [-5.57142857e-05]])
```

Reaction Forces

For every specified prescribed displacement we have one reaction force that we need to compute, once we have solved for the displacements that were not specified.

Given we have four specified displacements

```
*BOUNDARY
1 1 0
1 2 0
4 1 0
8 1 0
```

The ReactionForces:

```
matrix([[-1.00000000e+01],
        [ 7.10542736e-15],
        [-1.00000000e+01],
        [-4.00000000e+01]])
```

Correspond from top to bottom, index for index to the reaction forces computed for each specified `*BOUNDARY with the direction implied by direction specified under *BOUNDARY.`

`*BOUNDARY 1 1 0`
`Correspond to Reaction Force`
-1.00000000e+01, which acts in the x-direction

`*BOUNDARY 1 2 0`
`Correspond to Reaction Force`
7.10542736e-15, which acts in the y-direction

`*BOUNDARY 4 1 0`
`Correspond to Reaction Force`
-1.00000000e+01, which acts in the x-direction

`*BOUNDARY 8 1 0`
`Correspond to Reaction Force`
`-4.00000000e+01, which acts in the x-direction`

Force and moment equilibrium must hold i.e.

Sum of applied forces in x-direction + Sum of Reaction forces in x-direction must equal to 0

Sum of applied forces in y-direction + Sum of Reaction forces in y-direction must equal to 0

Moments of applied forces around point and moments of reaction forces around same point must equal to 0

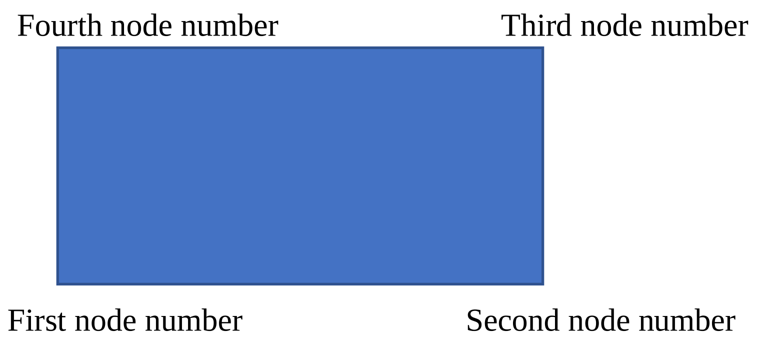
Example above: Reaction forces in x-direction = -10 – 10 – 40 = -60
Applied Forces in x-direction = 10 + 10 + 40 = 60
Sum of applied forces in x-direction + Sum of Reaction forces in x-direction = -60 + 60 = 0

Similarly for y-direction and moments.

Stress per element

The Stress per element is reported at the four corners of the element
The **element number – 1** corresponds to the **row index** in the Stress matrix, while each column corresponds to a node number of the element.

Under ***ELEMENT** , the first four nodes, for each element, defines the corners as follows:
***ELEMENT**
ELEMENT_NUMBER First node, Second node, Third node, Fourth node



Given VonMises Stress contains:
array([[53.3291665, 53.3291665, 53.3291665, 53.3291665]])

and our element definition is

```
*ELEMENT
1 1 2 3 4 5 6 7 8
```

```
We see that for element 1, give by row index 1-1 = 0, that the four stress
values in VonMises correspond to the defied node numbers 1-4.

Similarly for SXX, SYY and SXY
```