CM: 226

ME522

Advanced Finite Element Analysis

S. Jones

Spring 2016 - 17

*Creation of a 2D plane stress and strain finite element program.*

18 May 2017

Thaddeus Hughes

Dan Engstrom

Rose-Hulman Institute of Technology

Terre Haute, IN 47803

**Introduction**

In partial fulfillment of ME522, Advanced Finite Element Analysis, a 2D plane stress and plane strain finite element program was created. The program reads in DXF files to obtain model geometry, to which the user can apply loads and constraints based on assigned point and edge numbers.

The program is capable of applying both point and edge constraints as well as point and edge loads, pressures, and torques. The user enters linear, isotropic material properties (modulus of elasticity and Poisson’s ratio), and then can apply the thickness of their part. If desired, the mesh can be refined via a global face-sizing parameter.

After solving, the following outputs are made available: displacement (x, y, total), strain (x, y, shear), and stress (x, y, shear, von Mises). These can be overlaid with the mesh if the user so desires. There is also an option to view the original setup, in case the user wants to check constraint and loading conditions after solving.

The program has been validated using four different patch tests to ensure the accuracy of the stiffness matrix and forcing vector generation. These patch tests included a 2D rectangular element in tension, a cantilevered beam under concentrated loading, a cantilevered beam with an applied moment, and a thin walled pressure vessel. Additionally, a simulation of a fairly complicated gearbox plate has been performed and compared to ANSYS.

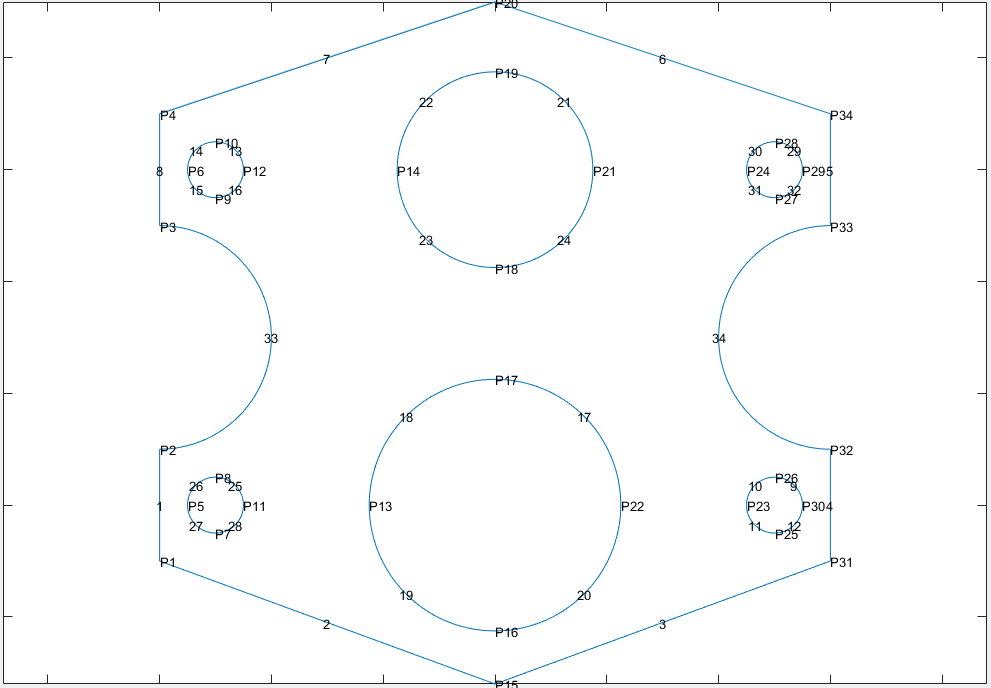
**Program Structure**

Our program follows the following steps (for the most part) in the following order:

1. Loading DXF geometry and translating it into MATLAB "decomposed geometry"
2. Meshing decomposed geometry
3. Formulating unconstrained stiffness matrix
4. Applying loads and constraints
5. Solving
6. Post-processing results

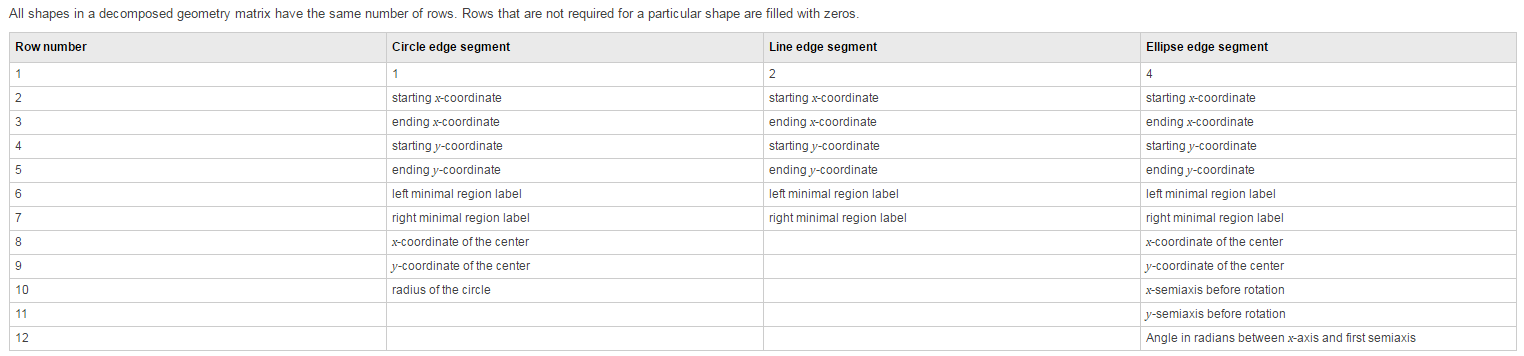
*i. Loading and translating geometry*

The DXF file format dates back to 1982 when it was first released by Autodesk as a means of moving data between AutoCAD and other programs, but has since become a widely-used standard for CAD geometry, since it is (relatively) easy to parse. Complex geometry in 3 dimensions can be created, including ellipses, parabolas, and splines. For this program, we only use three shapes: lines, circles, and arcs. Lines are described by their endpoints, circles by their center points and radii, and arcs by center point, radii, and start/end angles.



**Figure 1.** Example geometry capable of being processed, with edges and points labeled.

MATLAB includes a mesher (more on this later) which takes in "decomposed geometry". Decomposed geometry is simply a matrix with each column representing a different shape of the geometry. This also requires that circles and arcs be broken up into smaller chunks; a circle becomes four equally-sized arcs, split by quadrant. In this decomposed geometry, starting and ending coordinates are also specified, and calculating these so that the mesher recognizes connectivity between points proves difficult, as initmesh’s tolerance is higher than the accuracy of sin and cos functions. Attempts to fix and get around this (i.e. cloning nearby coordinates to be identical) proved to be ineffective. As such, support for arcs is limited to "nice" angles; that is, ones where sines and cosines can be computed with high accuracy.



**Figure 2.** Decomposed geometry standard, taken from MATLAB documentation on descg

After loading this geometry, it is given to the user with edges and points numbered. In the GUI, the user specifies material properties, mesh size, and commands for loads. After that, the program proceeds as normal. Like many FEA packages, after the solution is finished, the user may return to this step to modify their setup or load a new DXF file without changing other parameters of the simulation. Sadly, however, recognizing edges and renumbering them isn't available, so unlike in ANSYS where a modification of the geometry can be loaded without points of application being changed, you must re-specify which edges and points have loads applied to them.

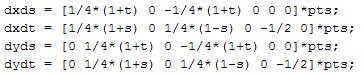
*ii. Meshing*

MATLAB includes a mesher for its PDE toolbox. This mesher, sadly, only supports triangular elements. This can be accessed a few different ways, but the simplest is initmesh. Aside from a decomposed geometry matrix, initmesh accepts one parameter: global sizing control. initmesh gives three output matrices: p (points), e (edges) and t (triangles). p is a 2xn matrix (where n is the number of points) containing x any y coordinates of all mesh points. e is a 6xm matrix (where m is the number of edges), where the first two rows contain the indices of the starting and ending points for the edges. t is a 4xl matrix (where l is the number of triangles), where the first three rows contain the indices of the edges.

MATLAB has some functions which can be used to improve this initial mesh. The first is jigglemesh, which does (somewhat) exactly what it says: it jiggles the points in the mesh to improve element quality. The other function is refinemesh, which can either cut all edges of triangles in half, or just the longest edge. This can be applied to specific triangles if used properly. Unfortunately, neither of these were utilized in this program, but could be added in the future to improve results without running into memory or computational time limits.

*iii. Stiffness matrices*

Generating an unconstrained stiffness matrix is not a terribly difficult process. After generating all triangles, we simply iterate through each one and compute its' stiffness matrix and then directly assemble it into the global stiffness matrix. Finding the derivatives of position with respect to the s and t isoparametric coordinates is actually done fairly easily and robustly with the following formulas:



2-point Gaussian quadrature is applied in both directions (making this a 4-point sampling) to perform the integration.

*iv. Load/constraint application and formulation*

Loads and constraints can be applied to points and/or edges depending on the type of load or constraint. In all cases, the following syntax is used:

<EDGES> <POINTS> ; <FUNCTION\_NAME> ; <ARGUMENT> <ARGUMENT> ...

***Examples:***

E1 E2 ; EDGE\_FORCE ; 1000 0

would apply an force of 1000 in the +x direction to edges 1 and 2.

P4 P2 ; POINT\_STRIKE ; 3

would apply a fixed support by an elimination method to points 2 and 4.

The first thing required to implement these is finding what points in the mesh correspond to the edges and points in the geometry- this in itself is difficult.

Tolerance is defined as the minimum of the mesh size and the total length of the geometry edge.

For geometry edges that are lines, the following conditions are assessed for all points in the mesh:

* Shortest distance from point to line must be within the tolerance
* The point coordinates must be within the bounding box of the line, with tolerance

For edges that are arcs, the following conditions are assessed for all points in the mesh:

* The point coordinates must be within the bounding box of the arc (defined by its start and end points), with tolerance. This may sound like a problem, but because of the decomposed geometry, arcs are never greater than 90 degrees, and so this bounding box is indeed correct.
* The distance from point to center of the arc must be within tolerance of the arc's radius.

After finding mesh points, the corresponding edges can be back-calculated by finding all mesh edges which only have indices for these points.

Here are all boundary conditions implemented, with a brief description of how the load is formulated:

EDGE\_FORCE ; <x force> <y force>

This begins by computing the total length of the edges the force is applied to. Then, all edges are iterated through, and a force of (where F is the force vector, l is the length of a mesh edge, and L is the total length of all mesh edges this load is applied to) is applied to each point of the edge. This creates a load that is uniformly applied to the surface, regardless of mesh sizing irregularities.

POINT\_FORCE ; <x force> <y force>

This takes all relevant points, and applies a load equal to the given force divided by the number of points to each point.

EDGE\_PRESSURE ; <pressure>

This iterates through all matching mesh edges and applies a load normal to the edge in direction, and equal in magnitude to (where P is the applied pressure, t is the material thickness, and l is the length of a mesh edge) to each point of the edge.

EDGE\_MOMENT ; <+ccw moment>

This begins by finding the centroid of all the edges this load will be applied to, then computes the moment of inertia (I; the sum of distances of edges to the centroid, weighted by the length of each edge). Then, all edges are iterated through, and each point on the edges are given a load in the x-direction equal to . Similarly, in the y direction, a load of is applied. Note the sign convention; this creates a positive-counter-clockwise moment. Load is formulated in this fashion to replicate the fashion in which bending moments and torques are generally applied; that is, the further away from the neutral axis (centroid), the larger the stress.

EDGE\_STRIKE ; <direction> and POINT\_STRIKE ; <direction>

These commands take the direction given and generate a "strikes" vector populated with the DOFs that should be removed with an elimination method. The direction is 1 for x direction, 2 for y direction, and 3 for both x and y direction. The rows and columns are not removed until all boundary conditions have been computed.

EDGE\_PENALTY ; <direction> and POINT\_PENALTY ; <direction>

These commands aim to accomplish the same as EDGE/POINT\_STRIKE, but utilize a penalty approach, applying a stiffness equal to the largest value, scaled by 106 in the stiffness matrix to each cell corresponding to a degree of freedom which is to be constrained. This method has not proved to be very robust.

*v. Solving*

After combining the loads, adding stiffness to the stiffness matrix, and removing DOFs from the force vector and stiffness matrix, solving is as simple as using MATLAB's \ operator to solve the system of equations . The q vector is then injected with zeros where DOFs were removed, in order to make post-processing easier.

*vi. Postprocessing*

Displaying results for deflection in x and y direction is as simple as pulling out every other value from the stiffness matrix. To find total deflection, use Pythagorean theorem. These results vary from node to node.

Due to the constant-strain nature of triangular elements, stress and strain vary from element to element, making computing these results fairly simple. Simply iterate through all elements and use cached values of B matrices from stiffness matrix generation to compute strain, then a cached D matrix to compute stress. von Mises stress can also be found with

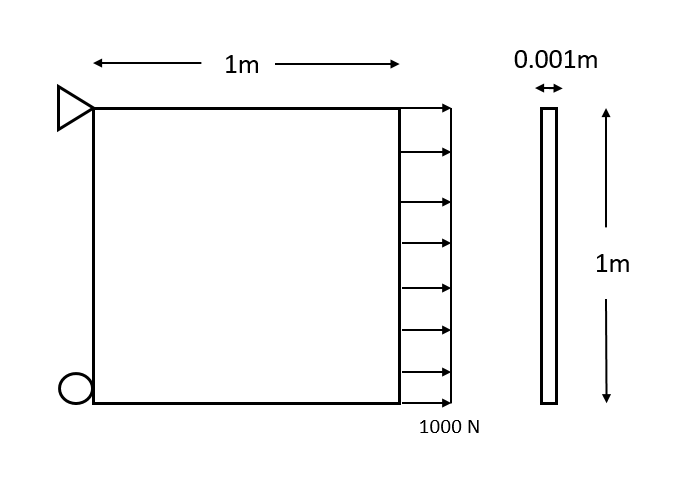
These results are then placed on a heatmap, with an option to turn off/on the mesh.

**Program Validation**

To validate the results of the program, four different patch tests were run -- plate in tension, cantilever beam experiencing concentrated loading, cantilever beam experiencing an external moment, and a thin-walled pressure vessel. All four were assumed to be steel: a linear, isotropic material. With these four simulations, the results from the four supported load types can be validated.

*i. Plate in tension*

For the first validation case, a plate was put in tension as shown in the below figure.



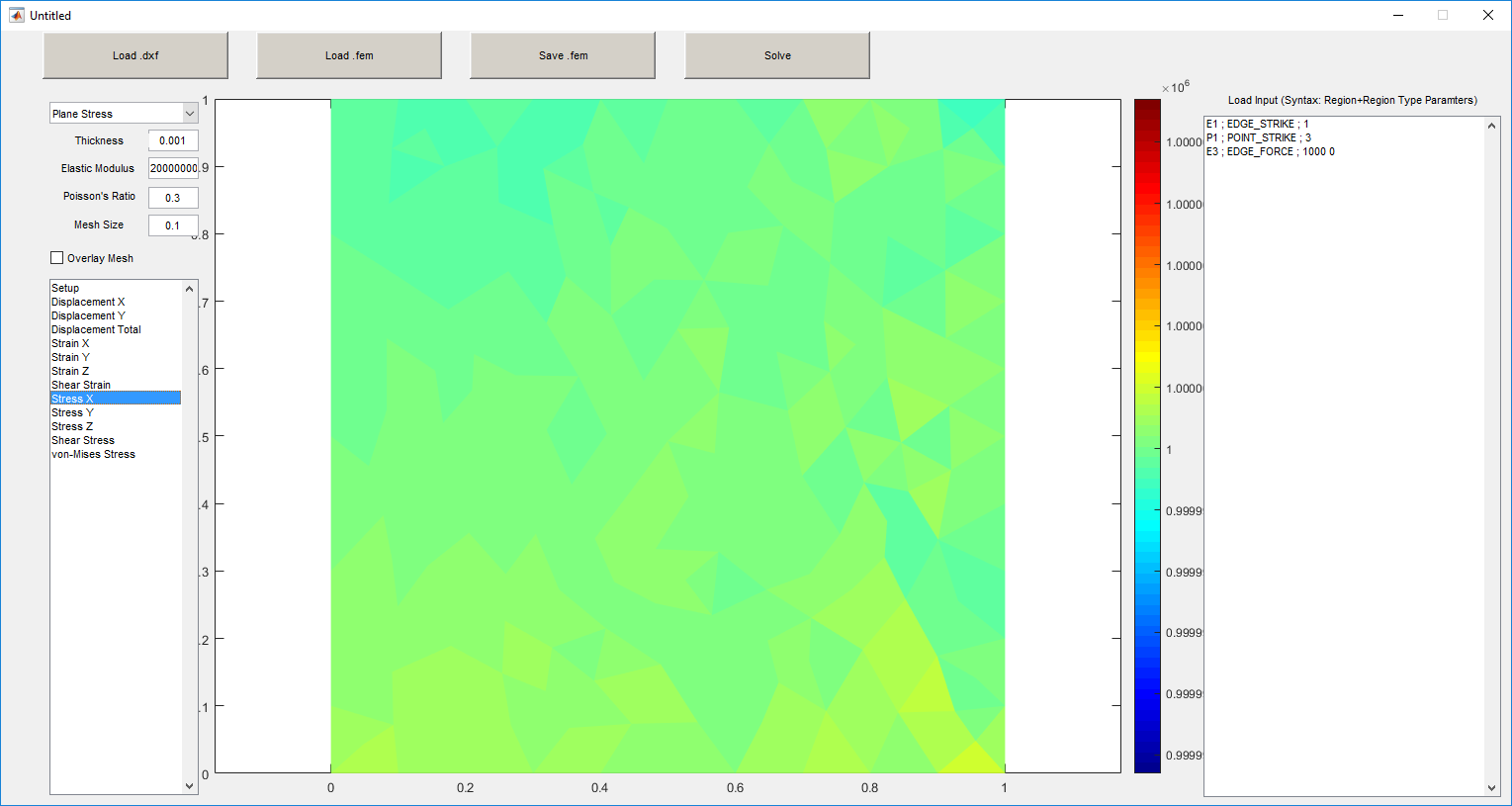
**Figure 3.** Plate in tension under 1000 N load

The analytic solution for the stress in the x-direction for this model can be obtained from

|  |  |  |
| --- | --- | --- |
|  |  | (1) |

where *σ* is stress, *F* is the applied force, and *A* is the cross-sectional area. Solving the above equation, the stress was found to be 1 MPa.

For the FE simulation, the left edge of the model was constrained in the x-direction, and the top-most point was fixed. Then an edge load of 1000 N was applied to the right edge of the model. This can be seen in the load and constraint panel to the right of the solution in Figure 4. The mesh was refined with a face-sizing of 0.1 m on the entire part.

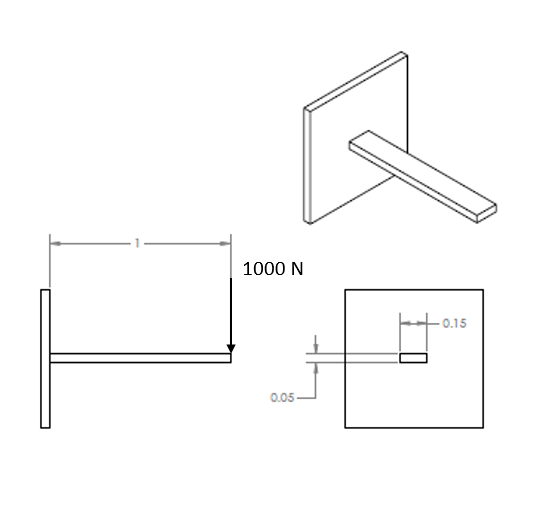


**Figure 4.** Plate loaded in tension, results for stress in the x-direction shown

As seen in the above figure, the plate is experiencing a stress in the x-direction of approximately 1e6 Pa (1MPa), just as the analytic solution predicted.

*ii. Cantilevered beam under point load*

For the second validation case, a concentrated load was applied to the end of a cantilevered beam. This will serve to validate the point load function. The setup used can be seen in the below figure.



**Figure 5.** Cantilevered beam experiencing point load

For the cantilevered beam, the displacement in the y-direction will be solved for analytically. This is done using Euler-Bernoulli beam theory.

In order to find the displacement, the moment of inertia of the beam is necessary. For a rectangular cross section, this can be calculated as

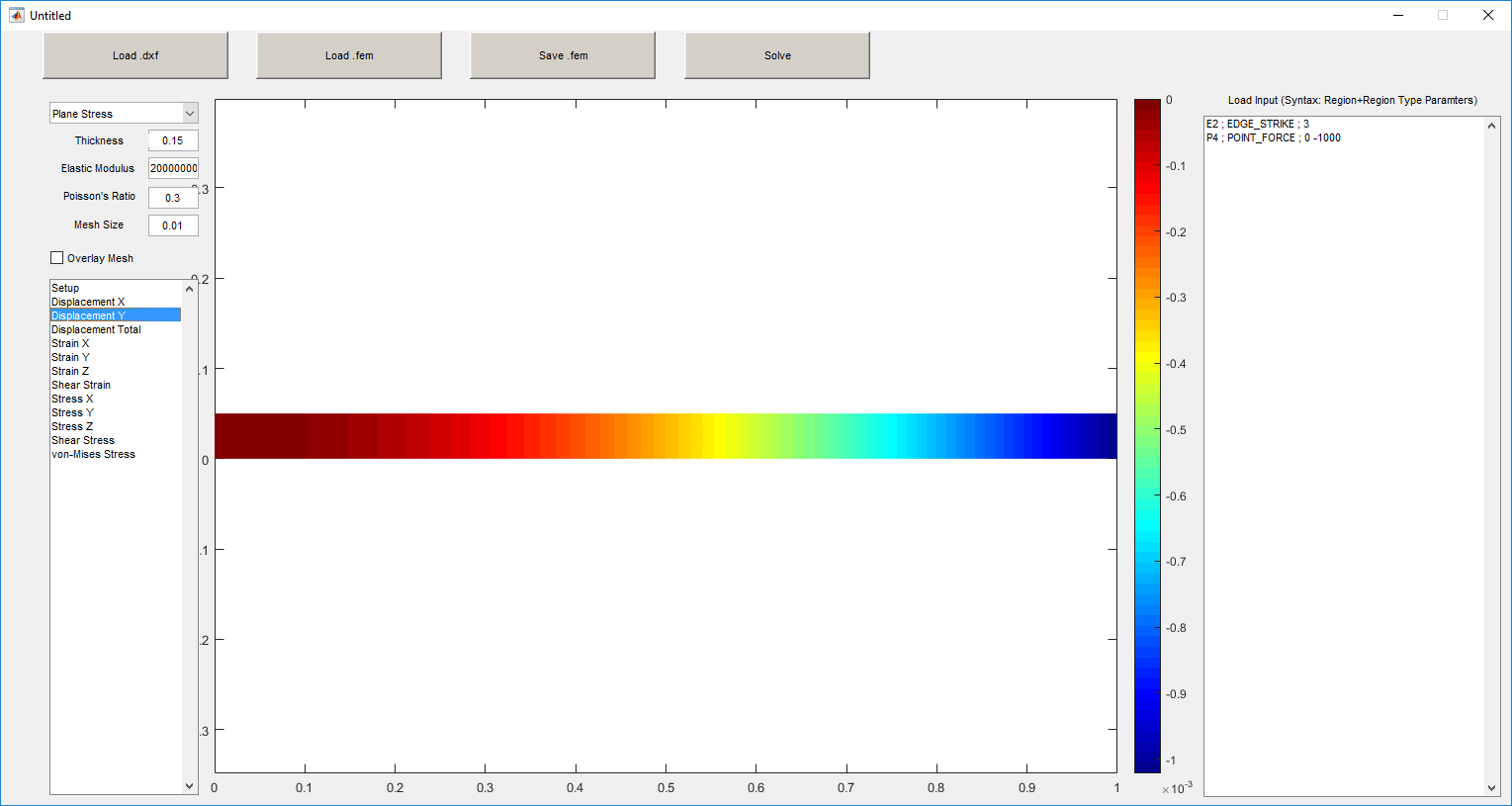
|  |  |  |
| --- | --- | --- |
|  |  | (2) |

where *I* is the moment of inertia, and *b* and *h* are the base and height of the beam, respectively.

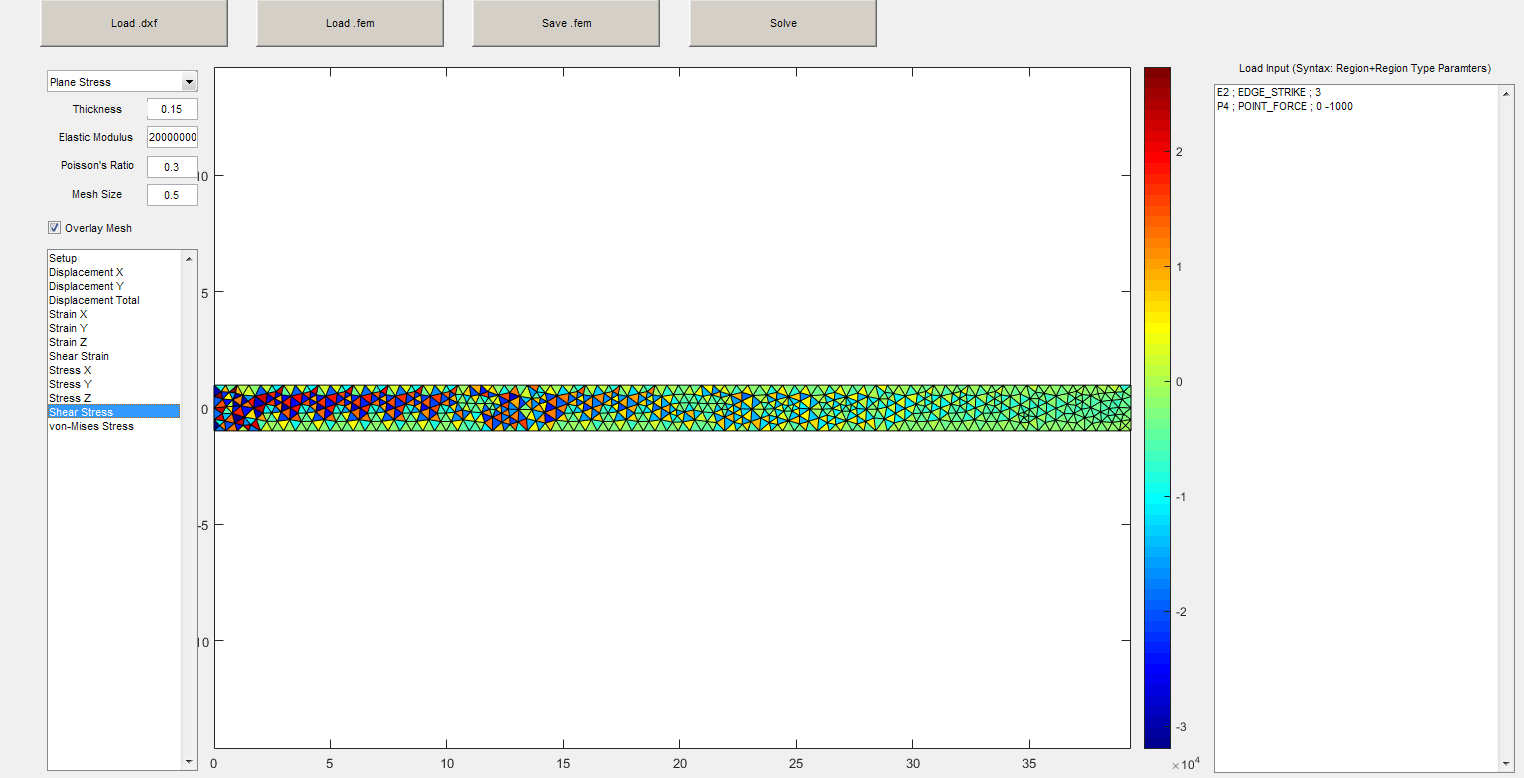
From this, the maximum beam deflection *ymax* can be found from the equation

|  |  |  |
| --- | --- | --- |
|  |  | (3) |

For the FE simulation, the left end of the beam is fixed in both the x- and y-directions, and a point load of 1000 N is applied in the negative y-direction on the top right node of the beam. These constraints and loads can be seen in Figure 6. Furthermore, the beam has a face-sizing of 0.01 m.



**Figure 6.** Cantilevered beam with point load applied, resultant y-displacement shown



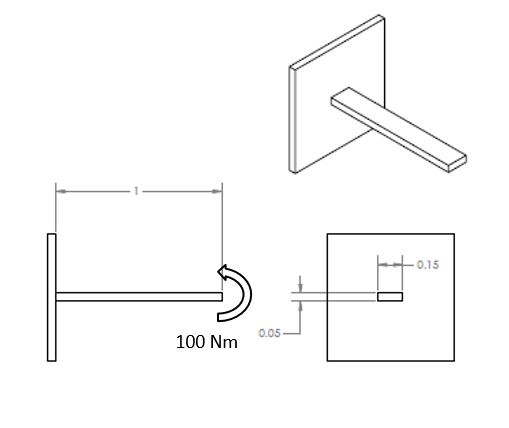
**Figure 7.** Cantilevered beam with point load applied, resultant shear stress shown

As seen in the above figure, the maximum displacement in the negative y-direction is around -1x10-3 m. This result matches the analytic hand calculations, and therefore, the patch test results are good.

For this simulation, the user can also see the effects of shear locking with the CST elements in the shear stress results. The shear stress near the root of the beam is much higher in magnitude near the center and alternates between high and low stress states. Analytically, we should expect the shear stress to be constant throughout with a stress concentration near the constrained end, but this shear-locking makes it so only the free end of the beam demonstrates this behavior. Use of quadrilateral elements, or at least elements in a better laid out mesh would reduce this effect.

*iii. Cantilevered beam experiencing applied moment*

For the third validation case, a moment was applied to the end of a cantilevered beam. The setup used can be seen in the following figure.



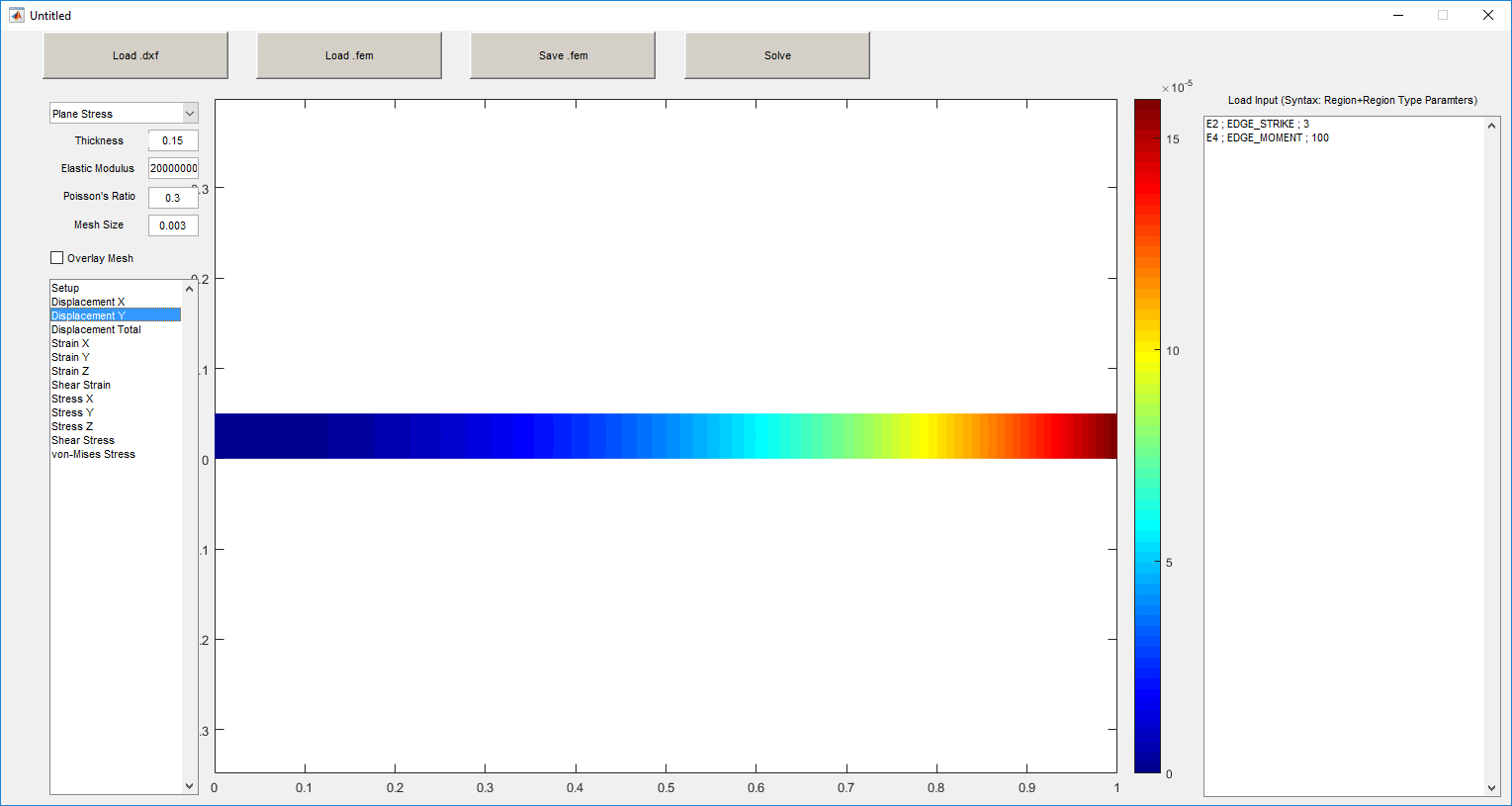
**Figure 8.** Cantilevered beam with applied moment

For this cantilevered beam, Euler-Bernoulli beam theory will once again be used. The beam dimensions are the same, and therefore, the moment of inertia calculated above will be used again.

Therefore, the deflection of the beam due to an applied moment can be found using

|  |  |  |
| --- | --- | --- |
|  |  | (4) |

For the FE simulation, the beam will again be fixed on the leftmost edge in both the x- and y-directions, and a moment of 100 Nm will be applied to the rightmost edge of the beam. The loading and constraints applied can be seen in Figure 9. Furthermore, a face-sizing of 0.01 m was applied to the entire face.

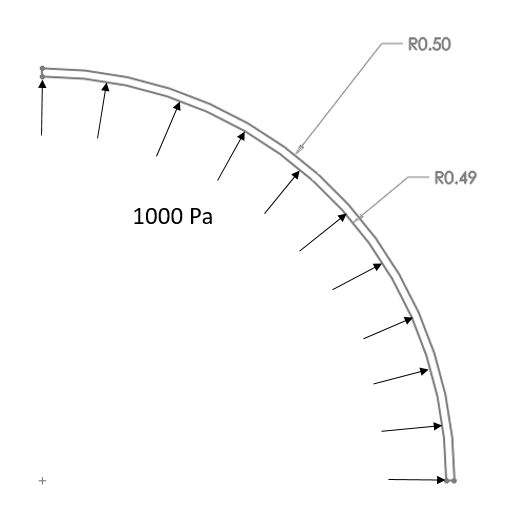


**Figure 9.** Cantilevered beam with applied moment, y-displacement results shown

As seen in the above figure, the maximum displacement in the y-direction is around 16x10-5 m. This result matches the analytic hand calculations, and therefore, the patch test results are good.

*iv. Thin walled pressure vessel*

For the fourth validation case, a ¼ model thin walled pressure vessel was analyzed. The setup can be seen in the following figure.



**Figure 10.** Thin walled pressure vessel with internal pressure

Since the thickness of the pressure vessel is less than 10 times its diameter, the thin walled assumption can be made. Therefore, the hoop stress the vessel will experience can be calculated using

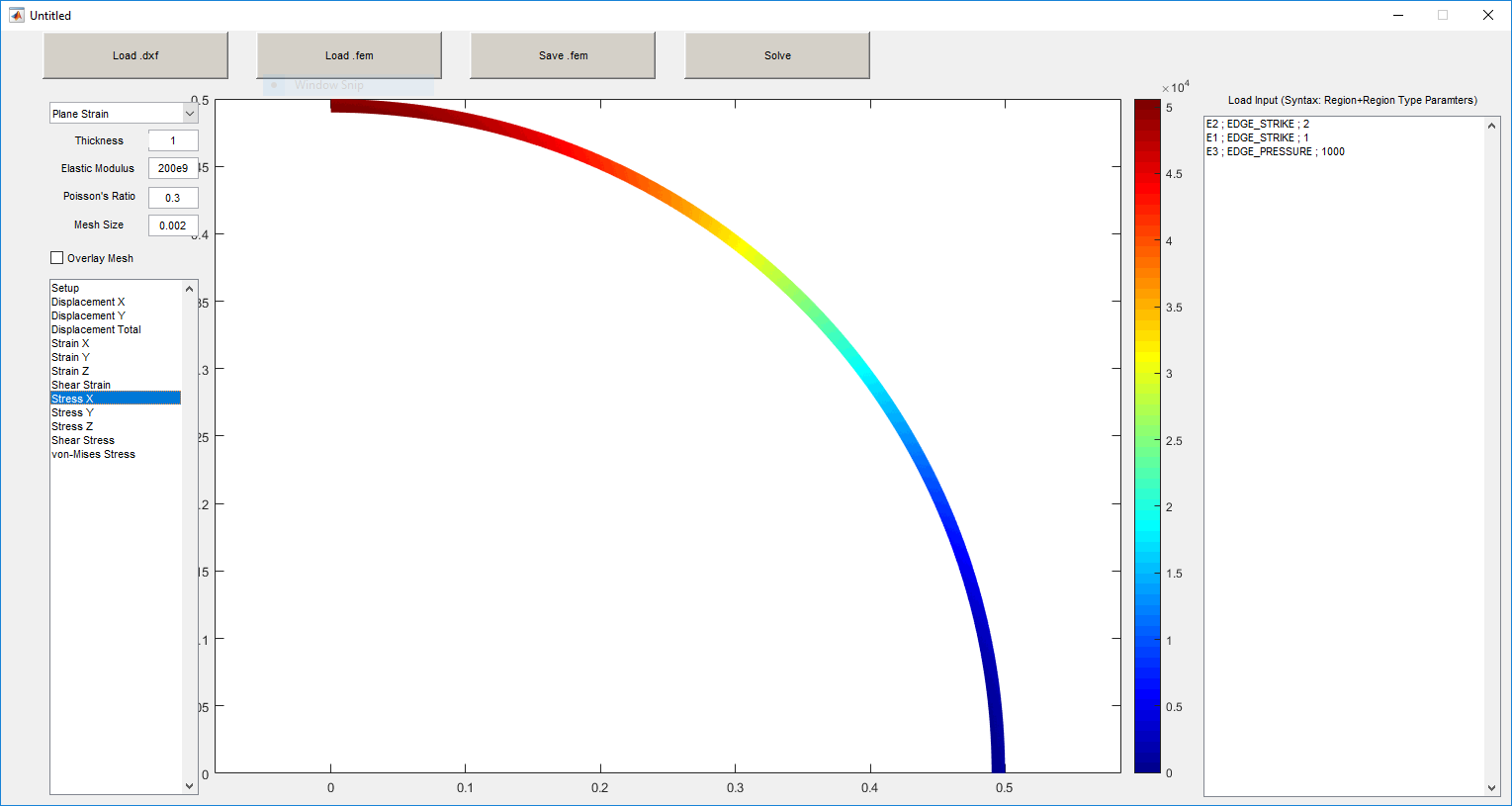
|  |  |  |
| --- | --- | --- |
|  |  | (5) |

where is the hoop stress, is the internal pressure, is the outside radius, and is the thickness.

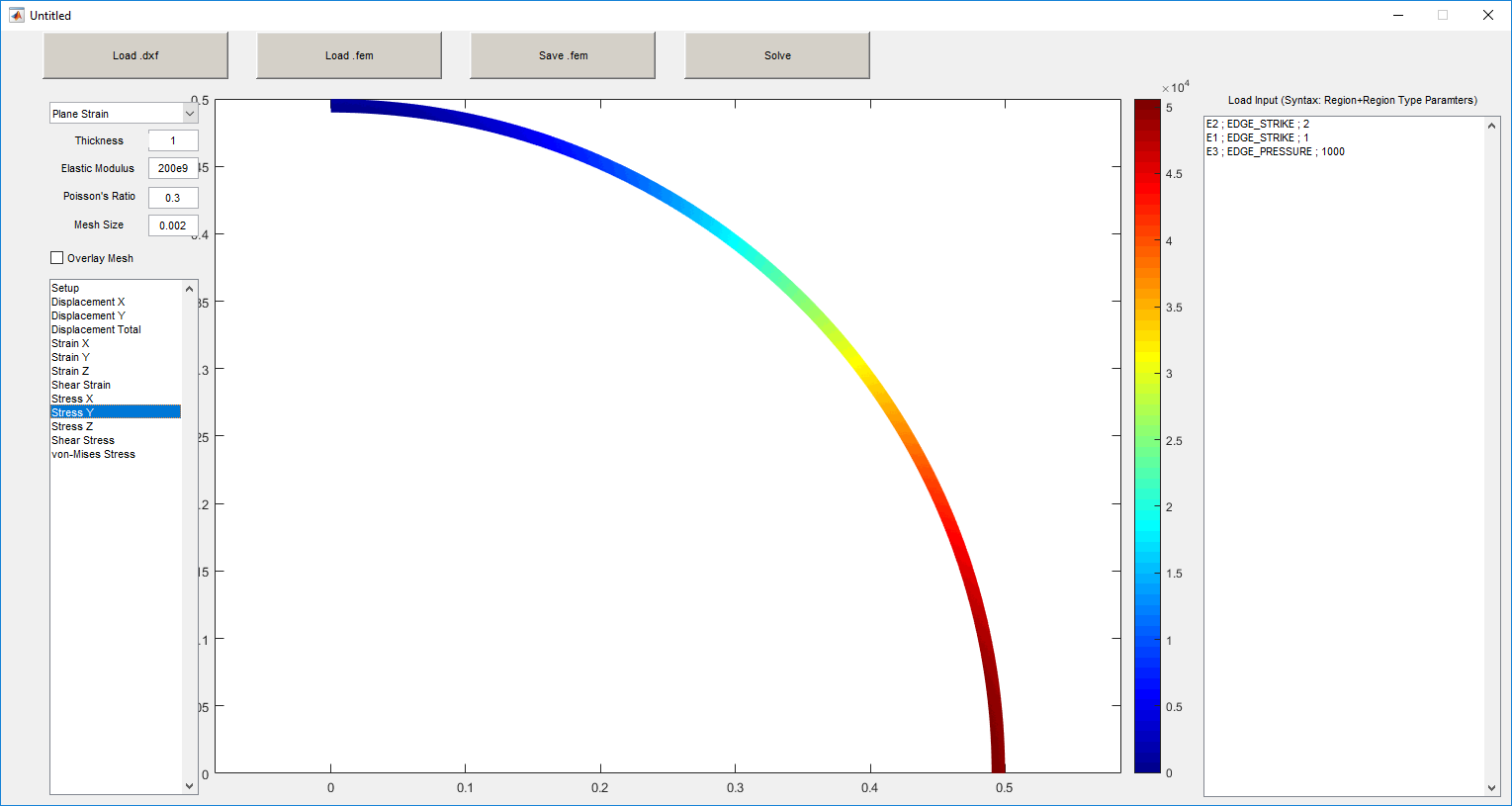
Therefore, from the analytic solution, a hoop stress of 50 kPa is expected.

Since the program only outputs stress in the x- and y-directions, hoop stress cannot be directly analyzed. However, the max stress in both the x- and y-directions should be equivalent to the hoop stress, since the load is applied normal to the cylinder.

For the FE simulation, the rightmost, vertical edge of the pressure vessel is constrained in the x-direction, while the lowest, horizontal edge is constrained in the y-direction (due to symmetry). An internal pressure of 1000 Pa is applied to the inner radius of the vessel. These loading conditions can be seen in Figures 11 and 12. Furthermore, a face-size mesh refinement of 0.002 m is applied to the entire surface.



**Figure 11.** Thin walled pressure vessel, stress in x-direction



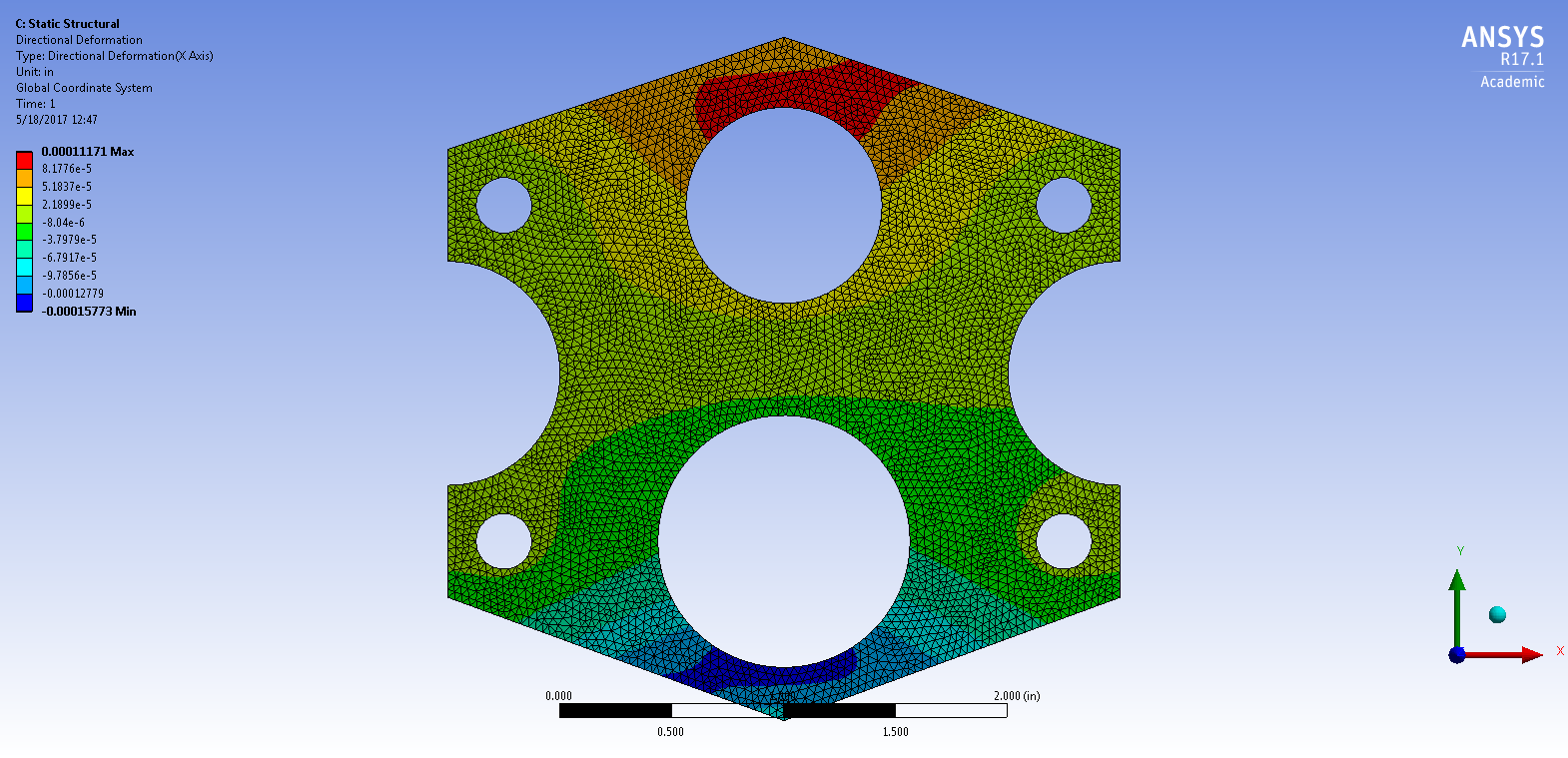
**Figure 12.** Thin walled pressure vessel, stress in y-direction

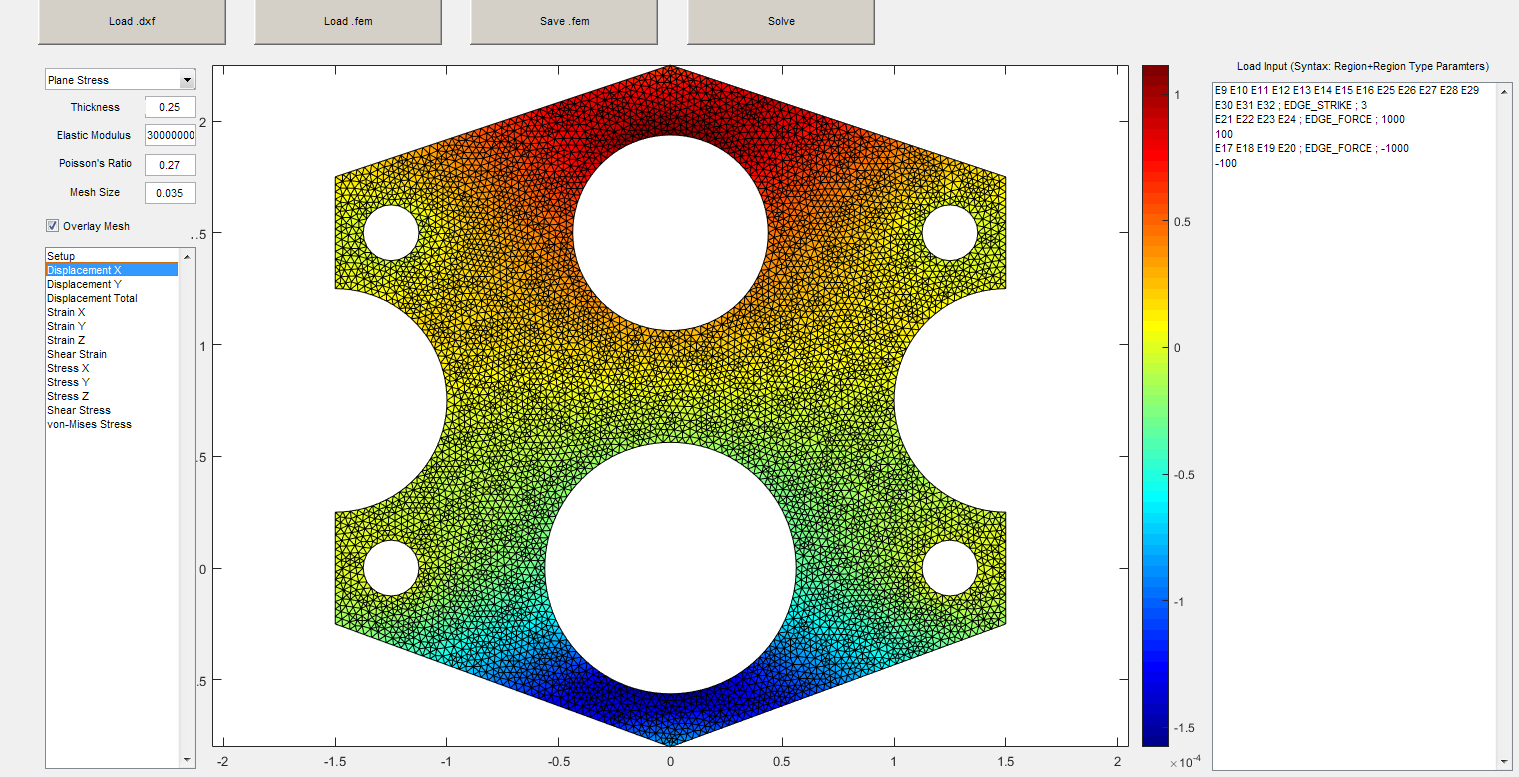
As seen in the two above figures, the maximum stress in both the x- and y-directions is around 5x104 Pa (or 50 kPa). This is what the analytic calculations predict, and therefore the patch test is valid.

A gearbox plate shown below in figure 13, with the four smaller bolt holes held fixed, a load of (1000,100) lbf applied to the top hole, a load of (-1000,-100) lbf applied to the bottom hole, is analyzed in both ANSYS and our FE program. ANSYS has been set to use the same mesh size, with triangular and linear elements. This results in the following results:

|  |  |  |  |
| --- | --- | --- | --- |
|  | **ANSYS** | **Our FE** | **% Error** |
| Max X displacement (in) | 1.12E-04 | 1.12E-04 | -0.098% |
| Min X displacement (in) | -1.58E-04 | -1.58E-04 | 0.098% |
| Max Y displacement (in) | 6.15E-05 | 6.12E-05 | -0.645% |
| Min Y displacement (in) | -9.65E-05 | -9.63E-05 | -0.202% |
| Max von Mises Stress (psi) | 5.56E+03 | 6.23E+03 | 12.154% |

Information about stresses should be taken with a grain of salt; this is not a converged mesh, so demonstrates significant mesh dependence around singularities, and our mesh is not identical to ANSYS’, just similar. However, seeing displacements of similar meshes match with less than 1% error provides high confidence in our algorithms.





**Figure 13.** X-displacement results for gearbox plate in ANSYS and our FE program.

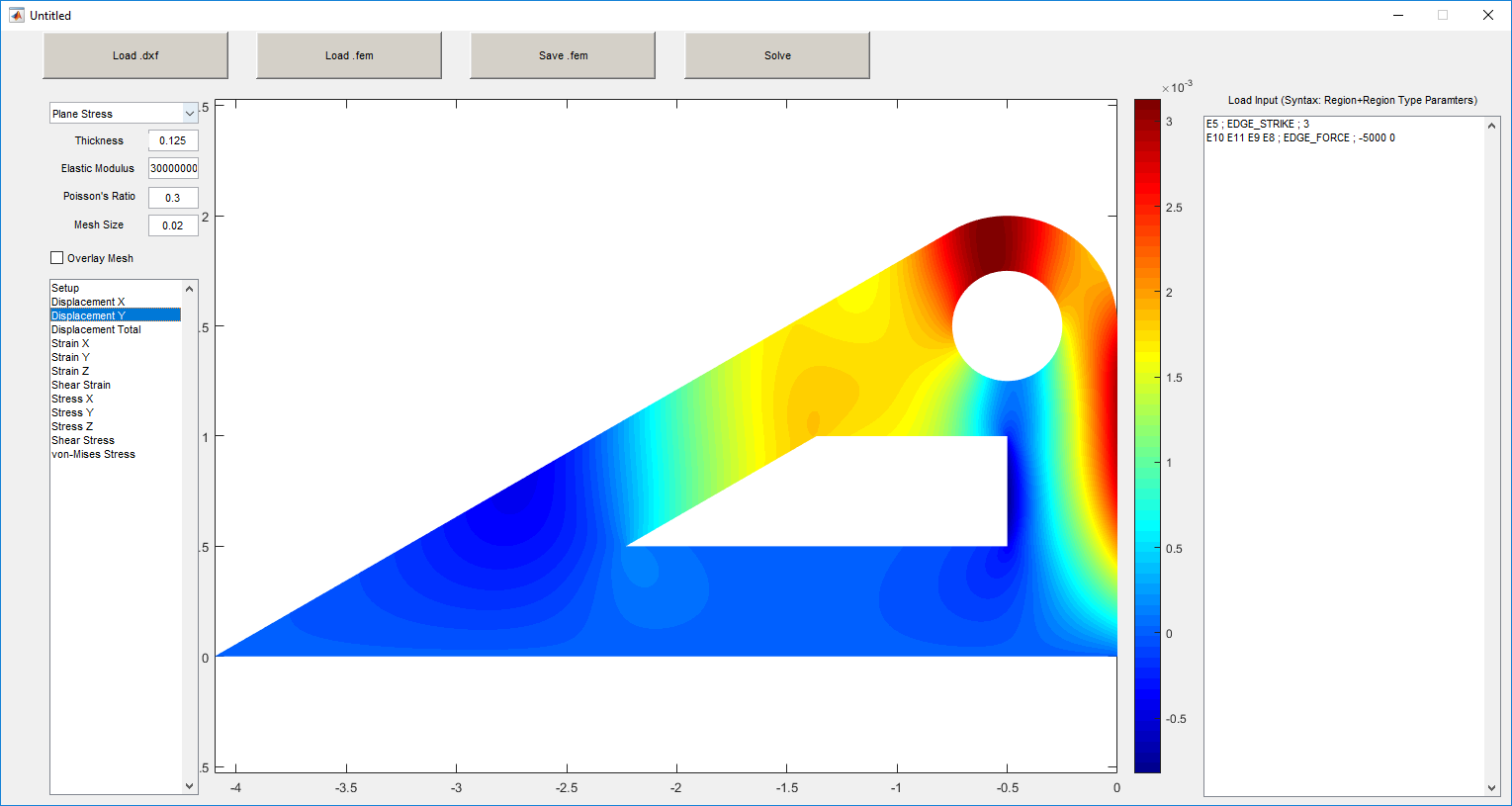
**Summary and Conclusions**

Some difficulties were encountered when implementing certain features of the solver.

One of these difficulties was creating a stable penalty function. The condition of the matrix (as evaluated with rcond) seemed to have a saddle point that was correlated to the penalty weight. Finding a penalty that worked for various geometries proved to be difficult. The function is written and works, but is very unstable. In order to obtain valid results, the user would have to change the penalty weight until an acceptable condition was found. Therefore, we reverted to using the elimination method to apply point and edge constraints for stability.

Furthermore, torques proved to be more difficult to implement than originally thought. The most confusing step was first trying to determine what a distributed torque meant, physically. After rethinking statics and materials fundamentals, this made sense and the algorithm fell into place.

Other than these two main difficulties, the program appears to be stable for most geometries tested. All four patch tests match their analytic solution, and more complex geometries (such as the tab seen below) can be solved. A variety of different loading and constraint combinations are possible, and most results vital to a user can be obtained.



**Figure 14.** Y-displacement of bracket

To increase the program’s function, we could rewrite a meshing algorithm to fix tolerance stack ups with arcs, as well as implement quadrilateral meshing.