MECH 151L: Finite Element Theory and Applications
Lab 4: Adaptive Mesh Refinement

German Markaryan

December 14, 2025

Abstract

The goal of this experiment is to investigate how adaptive mesh refinement can improve finite element analysis. There are 6 experiment setups, including uniform and adaptive mesh with porosity of 3% and aspect ratios of 1, 20, and 40. Results show that adaptive mesh refinement helps to create a very dense mesh in stress concentration and hence increases the accuracy of the analysis. The advantage of adaptive mesh increases with the increase in the magnitude of surface shape change. It is also shown that adaptive mesh refinement does not provide an advantage for the analysis of the effective Poisson's ratio. Overall, adaptive mesh refinement is a more effective way to increase the accuracy of the finite element analysis without increasing the computational needs.

1. Introduction

Mesh is a fundamental part of the finite element analysis that influences the results of the simulation and hence its accuracy in predicting the physical behavior of the part in the real world. Quality mesh is the one that yields reliable output results, and one way to make sure it happens is to check the convergence of the simulation.

Mesh density is one of the most important parameters that can be changed to make the simulation more accurate. The only problem is that it comes at the cost of the computing cost. For the bodies that have a uniform shape, such as circles and spheres, a uniform increase in the density of the mesh will be an efficient way to make the simulation more accurate.

However, the real world is not geometry perfect and often contains sharp angles, holes, or other irregular surfaces that require very high element numbers at stress concentration in order to yield meaningful results. The usual strategy to simply decrease the seed element is inefficient because most of the elements that will be added are going to be far from the point of interest.

One of the solutions to deal with it is an adaptive mesh refinement, and Abacus CAE has its own algorithms to accomplish it. This allows the mesh to be very dense near the point of interest and low-density in regions of no interest. This helps to keep a lower number of elements while getting better results, which saves a lot of computational power.

Since adaptive mesh refinement is done automatically, the algorithm must know and show if the change makes the simulation more accurate or not. ENDENERI and MISESERI error indicators are used to examine the iteration on the amount of error, and by plotting those errors for each iteration, it becomes clear if the change makes the simulation more accurate or less.

The goal of this experiment is to understand the difference between adaptive and uniform meshes. There are three simulation setups with aspect ratios of 1, 20, and 40. The specimen is a square with multiple holes and a porosity of 3%.

2. Results and Discussion

The first experiment is done with aspect ratio of 1 and seed size of 0.045. Figure 1 shows the Von-Mises stress for the uniform mesh. Figure 2 shows the same output but for the mesh with adaptive mesh refinement. Is is clear that the mesh near the stress concentration is much denser however the mesh density surrounding the hole is less dense than the uniform density shown on Figure 1.

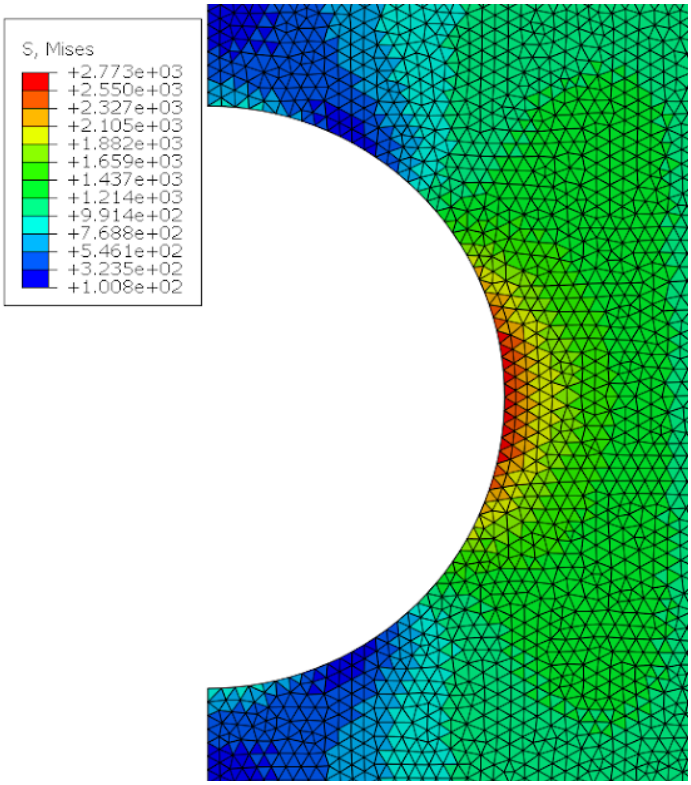

Figure 1: Von-Mises stress for the uniform mesh. Averaging is off. Mesh size = 0.045. Aspect ratio = 1.

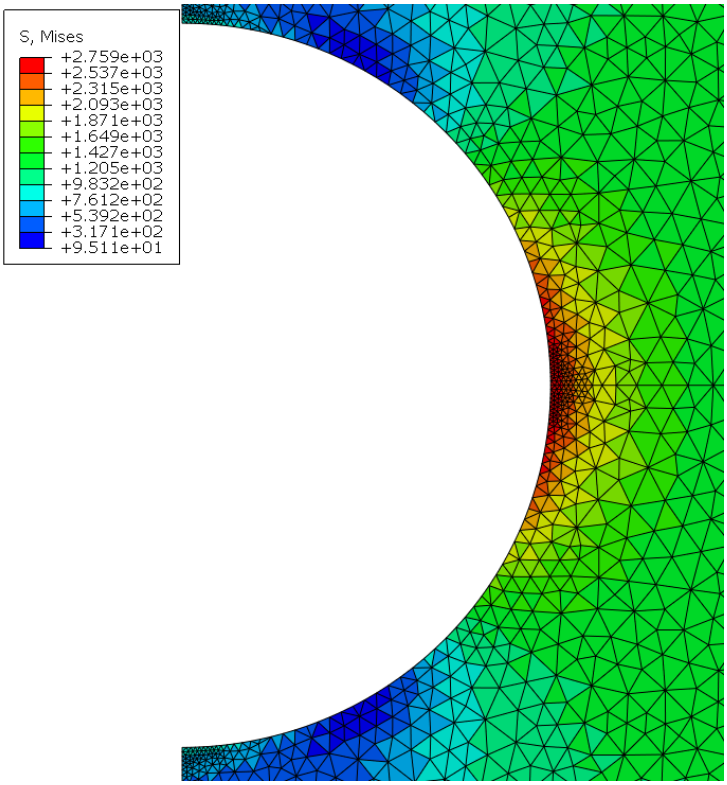

Figure 2: Von-Mises stress for the adaptive mesh refinement. Averaging is off. Mesh size = 0.045. Aspect ratio = 1.

Adaptivity Plot for this experiment is shown in Figure 3, and it shows that only two iterations were enough to see substantial progress in error reduction.

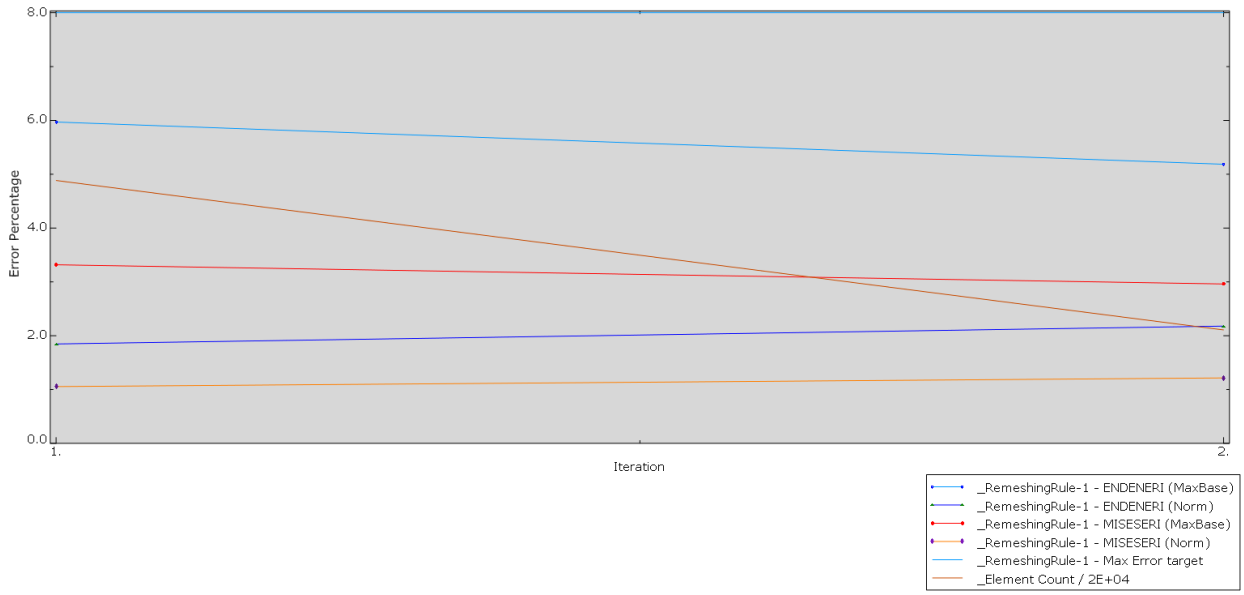

Figure 3: Adaptivity Plot for mesh size = 0.045 and spect ratio = 1.

The following observation is also true for the experiments with aspect ratios of 20 and 40 shown in Figures 4, 5, 6, 7, and 8. Adaptivity Plot for experiment with aspect ratio of 20 shows a much better error reduction as shown in Figure 9 due to adaptive mesh refinement that can be explained by a much sudden surface change due to higher aspect ratio, which means stress concentration is going to be on a smaller area. This means that the aspect ratio of 40 must show even better error reduction due to adaptive mesh refinement, and that holds true as shown in Figure 10.

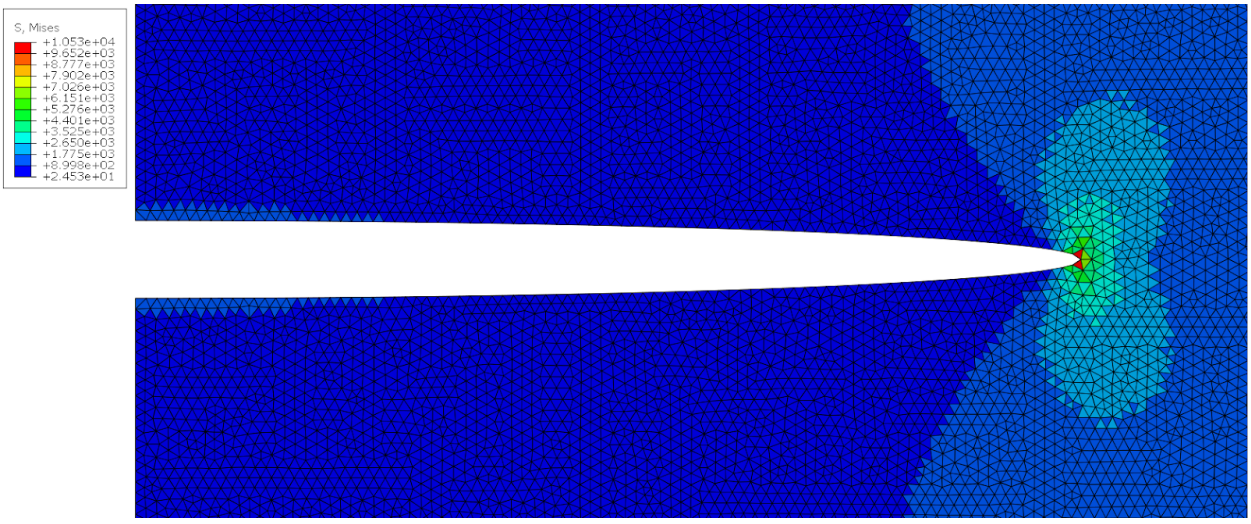

Figure 4: Von-Mises stress for the uniform mesh. Averaging is off. Mesh size = 0.05. Aspect ratio = 20.

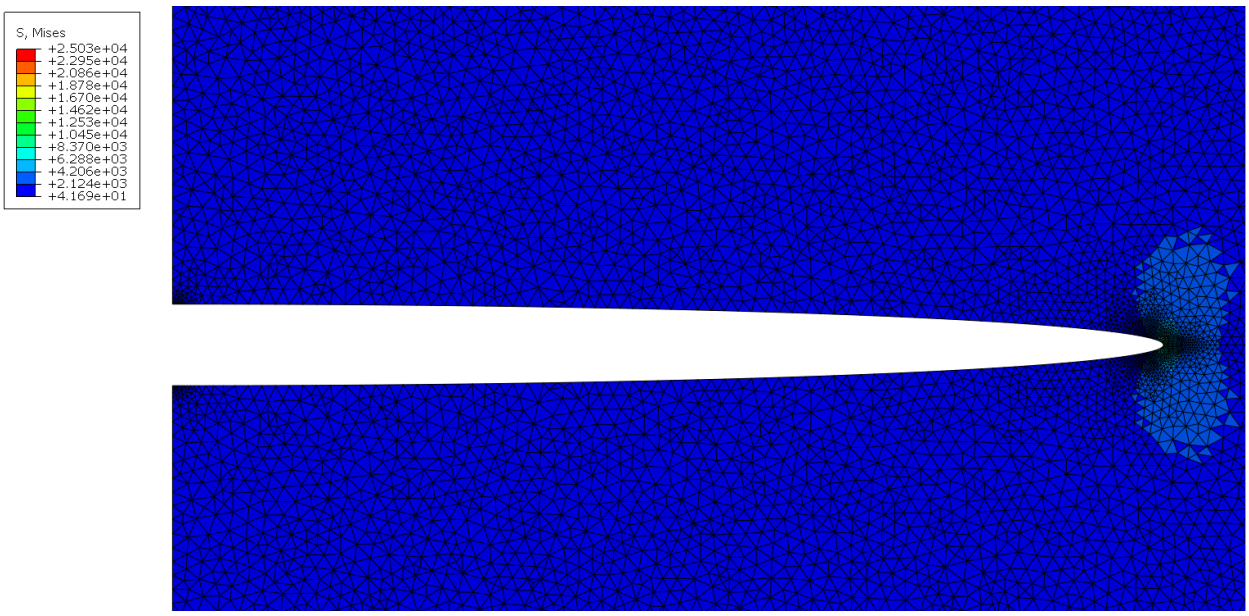

Figure 5: Von-Mises stress for the adaptive mesh refinement. Averaging is off. Mesh size = 0.05. Aspect ratio = 20.

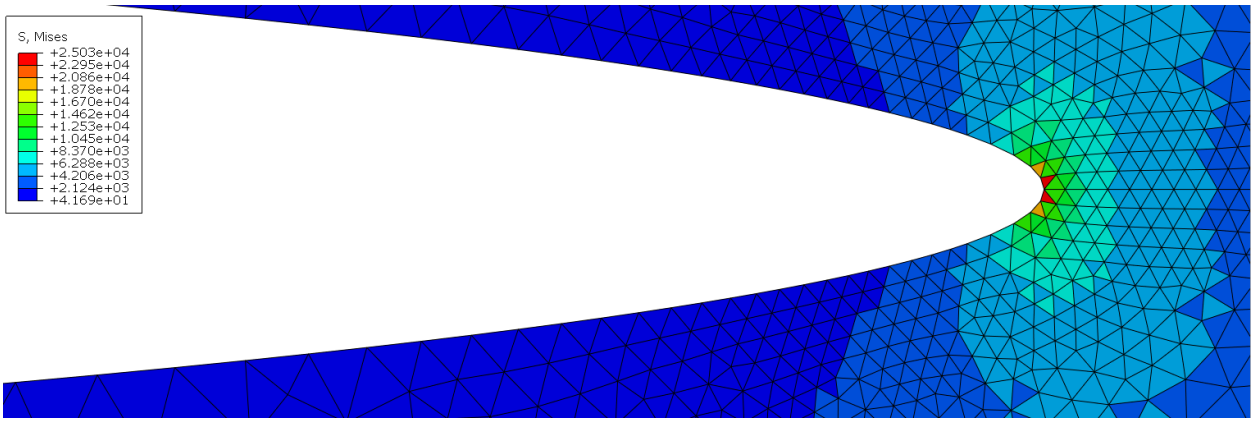

Figure 6: Von-Mises stress for the adaptive mesh refinement. Averaging is off. Mesh size = 0.05. Aspect ratio = 20. Zoomed in.

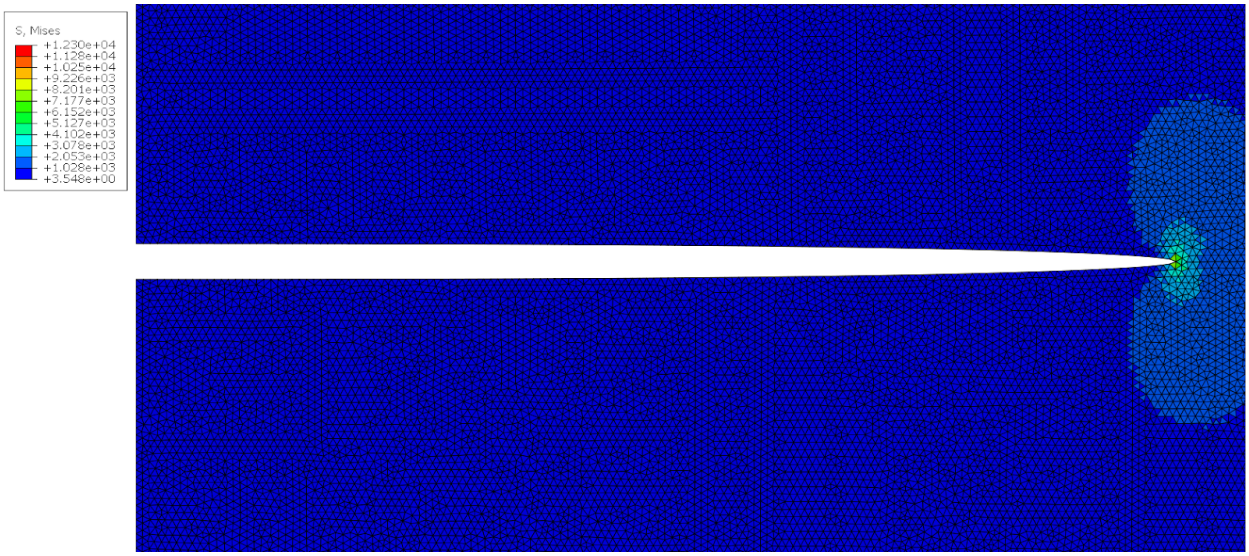

Figure 7: Von-Mises stress for the uniform mesh. Averaging is off. Mesh size = 0.04. Aspect ratio = 40.

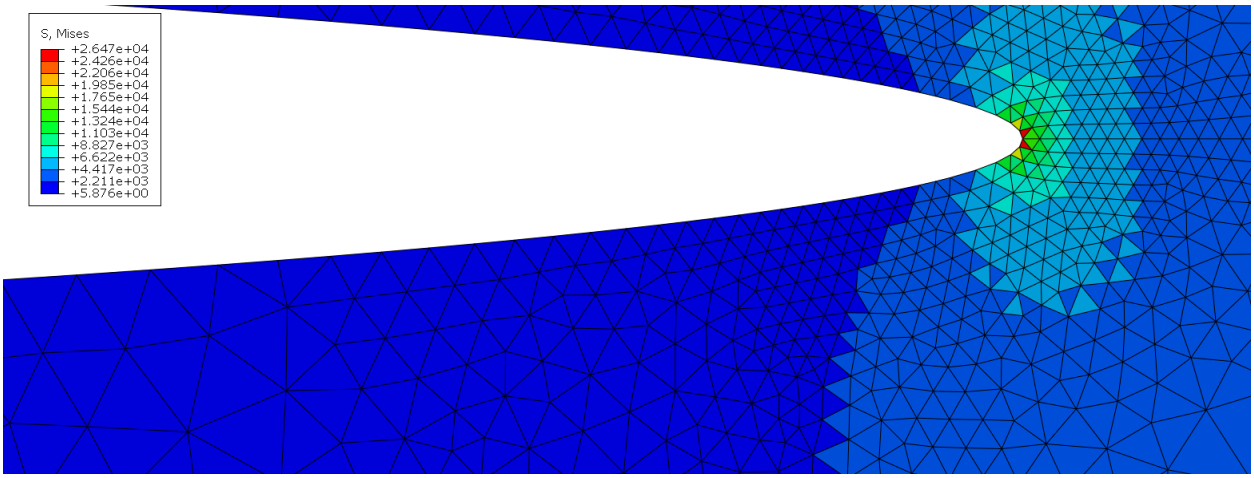

Figure 8: Von-Mises stress for the adaptive mesh refinement. Averaging is off. Mesh size = 0.04. Aspect ratio = 40.

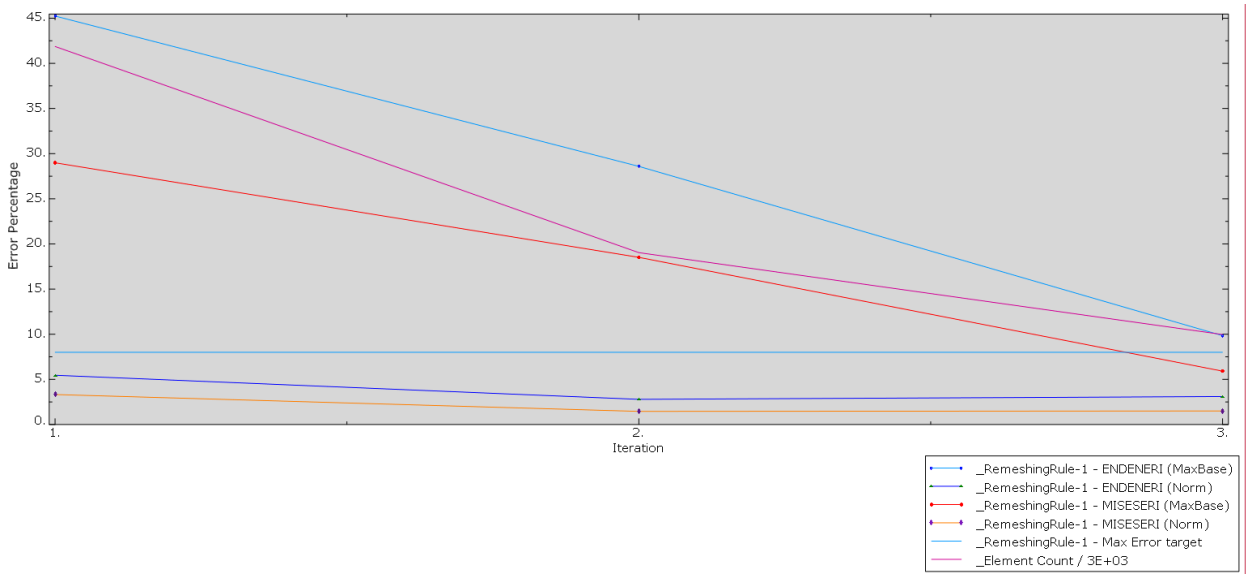

Figure 9: Adaptivity Plot for mesh size = 0.05 and spect ratio = 20.

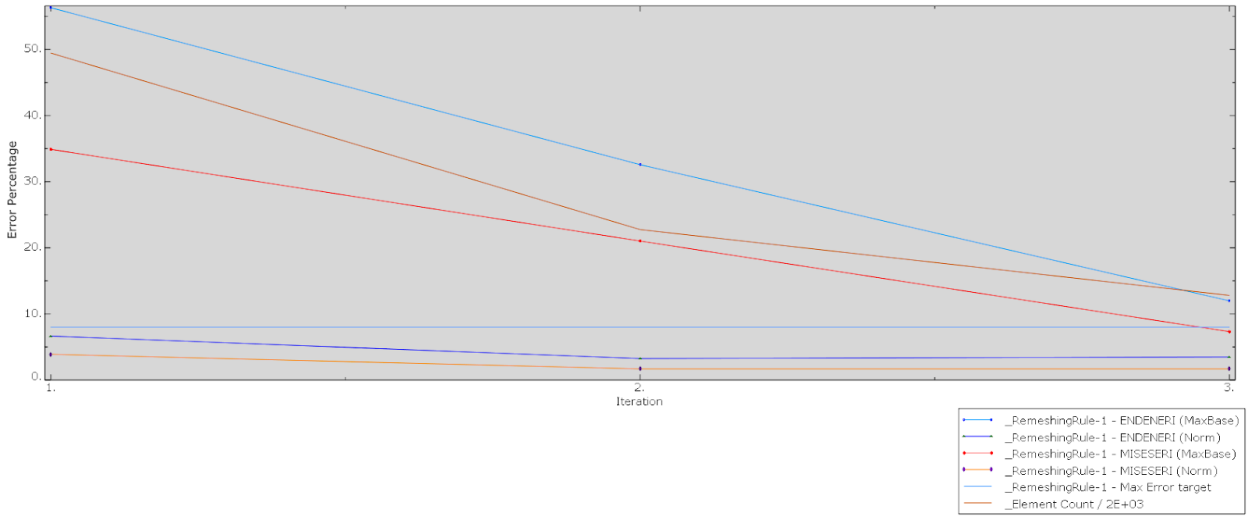

Figure 10: Adaptivity Plot for mesh size = 0.04 and spect ratio = 40.

Now, as we know that adaptive mesh refinement is working as intended, it's time to analyze how effective Poisson's ratio reacts to the adaptive mesh refinement in comparison to a uniform mesh. Using the strain from each experiment, Poisson's ratio was calculated, and the values are shown in Table 1. The experiment will also provide insights into how the maximum Von-Mises stresses and unique nodal Von-Mises stresses differ between the two mesh types. UM in Table 1 means uniform mesh, and AMR means adaptively refined mesh.

Table 1: Value for the Von-Mises stresses and effective Poisson's Ratio for all 6 experiments.

	Von-Mises, AVG - OFF	Von-Mises, Integration Point	Effective Poisson's Ratio
UM for AR=1 & Size = 0.045	2734.96	2772.74	0.30
AMR for AR=1 & Size = 0.045	2759.00	2759.38	0.30
UM for AR=20 & Size = 0.05	10384.60	10527.40	-0.06
AMR for AR=20 & Size = 0.05	25026.90	25027.40	-0.06
UM for AR=40 & Size = 0.04	7783.10	12300.10	-0.53
AMR for AR=40 & Size = 0.04	26470.10	26470.30	-0.53

Table 1 values show that Poisson's ratio does not change with the type of the mesh, and this makes sense; hence, the general deformation is not highly sensitive to mesh quality, and hence, mesh refinement does not make big changes in strain values. However, Von-Mises stress values do change depending on the type of mesh, and the higher the aspect ratio, the higher the difference between the two types of mesh is. This means that mesh refinement for the most sensitive specimen shapes indeed helps significantly in terms of accuracy.

3. Conclusion

This experiment confirmed that adaptive mesh refinement is highly effective for objects with a sudden change in surface shape. Adaptive mesh is also a much more effective way of applying dense mesh at points with high stress concentration since it allows more accurate results for the same mesh size, and hence it helps to save computational power. As expected, the effective Poisson's ratio is not sensitive to adaptive mesh refinement since the whole part deformation does not require a dense mesh at points of high stress concentrations.

Appendix

1. Code for adaptive mesh refinement:

```
#
# GERMAN!!! MARKARYAN!! YES! LETS GOOOOOOOOOOOOOO LAST LAB HELL YEAH!!!!
# 11/18/2025
#
# MECH 151L: Lab 4
# Biaxial load on quarter piece of RVE for Adaptive Mesh Refinement
#

Mdb()
pathName = "Z:/dcengr/My Documents/MECH151L/Lab 4/"
os.chdir(pathName)

# Includes
from part import *
from material import *
from section import *
from assembly import *
from step import *
from interaction import *
from load import *
from mesh import *
from job import *
from sketch import *
from visualization import *
from connectorBehavior import *
from array import *
from math import *
from shutil import *
import odbAccess

session.journalOptions.setValues(replayGeometry=COORDINATE,recoverGeometry=COORDINATE)
#
```

```
# Rename model
--------------------------------------------------------------------------------
modelName = 'model_2DVoidPlate'
mdb.models.changeKey(fromName='Model-1', toName=modelName)
#
--------------------------------------------------------------------------------
----


# Material properties
--------------------------------------------------------------------------------
YoungsMod = 200e3   # Young's modulus (in MPa)
PoissonRatio = 0.3
#
--------------------------------------------------------------------------------
----



# Geometric properties
--------------------------------------------------------------------------------
center_to_center = 10       # center to denter distance for the holes (in mm)
porosity = 0.03                 # porosity
aspect_ratio = 1.0              # ratio between major and minor axis for the holes
thickness = 0.0                     # thickness of the plates
seed_mesh = 0.04                    # seed-mesh (in mm)


width_plate = 2.0*center_to_center;       # width of plate
height_plate = 2.0*center_to_center;      # height of plate
area_plate = width_plate*height_plate;   # area of the plate
area_hole = area_plate*porosity/4.0;      # area of each void
minor_axis_hole = sqrt(area_hole/(pi*aspect_ratio));    # minor axis of each void
major_axis_hole = minor_axis_hole*aspect_ratio;             # major axis of each void


subPath = pathName + "AMR/" + "P" + str(int(porosity*100)) + "_AR" + str(aspect_ratio)
+  "/"
if not os.path.exists(subPath):
    os.makedirs(subPath)
os.chdir(subPath)
#
--------------------------------------------------------------------------------
----
```

```
# Create parts and instances for plate
----------------------------------------------------------------
partName='part_Plate'

#matrix
mdb.models[modelName].ConstrainedSketch(name='__profile__', sheetSize=200.0)
mdb.models[modelName].sketches['__profile__'].rectangle(point1=(0.0, height_plate/
2.0),
    point2=(width_plate/2.0, height_plate))
mdb.models[modelName].Part(dimensionality=TWO_D_PLANAR,
name='Matrix',type=DEFORMABLE_BODY)
mdb.models[modelName].parts['Matrix'].BaseShell(sketch=mdb.models[modelName].s
ketches['__profile__'])
del mdb.models[modelName].sketches['__profile__']

#void1
mdb.models[modelName].ConstrainedSketch(name='__profile__', sheetSize=200.0)
mdb.models[modelName].sketches['__profile__'].EllipseByCenterPerimeter(
    axisPoint1=(minor_axis_hole, 0.0), axisPoint2=(0.0, major_axis_hole), center=(0.0,
0.0))
mdb.models[modelName].Part(dimensionality=TWO_D_PLANAR,
name='Void1',type=DEFORMABLE_BODY)
mdb.models[modelName].parts['Void1'].BaseShell(sketch=mdb.models[modelName].s
ketches['__profile__'])
del mdb.models[modelName].sketches['__profile__']

#void2
mdb.models[modelName].ConstrainedSketch(name='__profile__', sheetSize=200.0)
mdb.models[modelName].sketches['__profile__'].EllipseByCenterPerimeter(
    axisPoint1=(major_axis_hole, 0.0), axisPoint2=(0.0, minor_axis_hole), center=(0.0,
0.0))
mdb.models[modelName].Part(dimensionality=TWO_D_PLANAR,
name='Void2',type=DEFORMABLE_BODY)
mdb.models[modelName].parts['Void2'].BaseShell(sketch=mdb.models[modelName].s
ketches['__profile__'])
del mdb.models[modelName].sketches['__profile__']

#Assembly
#matrix
mdb.models[modelName].rootAssembly.DatumCsysByDefault(CARTESIAN)
mdb.models[modelName].rootAssembly.Instance(dependent=ON, name=
    'Matrix-1', part=mdb.models[modelName].parts['Matrix'])

#Void 1C
mdb.models[modelName].rootAssembly.Instance(dependent=ON, name=
    'Void1-3', part=mdb.models[modelName].parts['Void1'])
```

```
mdb.models[modelName].rootAssembly.translate(instanceList=('Void1-3', ),
vector=(center_to_center, center_to_center, 0.0))
#Void 1D
mdb.models[modelName].rootAssembly.Instance(dependent=ON, name=
    'Void1-4', part=mdb.models[modelName].parts['Void1'])
mdb.models[modelName].rootAssembly.translate(instanceList=('Void1-4', ),
vector=(0.0, 2.0*center_to_center, 0.0))

#Void 2A
mdb.models[modelName].rootAssembly.Instance(dependent=ON, name=
    'Void2-1', part=mdb.models[modelName].parts['Void2'])
mdb.models[modelName].rootAssembly.translate(instanceList=('Void2-1', ),
vector=(0.0, center_to_center, 0.0))

#Void 2D
mdb.models[modelName].rootAssembly.Instance(dependent=ON, name=
    'Void2-4', part=mdb.models[modelName].parts['Void2'])
mdb.models[modelName].rootAssembly.translate(instanceList=('Void2-4', ),
vector=(center_to_center, 2.0*center_to_center, 0.0))

#cut
mdb.models[modelName].rootAssembly.InstanceFromBooleanCut(name=partName,
      instanceToBeCut=mdb.models[modelName].rootAssembly.instances['Matrix-1'],
       cuttingInstances=(
       mdb.models[modelName].rootAssembly.instances['Void1-3'],
       mdb.models[modelName].rootAssembly.instances['Void1-4'],
       mdb.models[modelName].rootAssembly.instances['Void2-1'],
       mdb.models[modelName].rootAssembly.instances['Void2-4'],
       ),originalInstances=SUPPRESS)


instName = 'inst_2DVoidPlate'
mdb.models[modelName].rootAssembly.features.changeKey(fromName=
    'part_Plate-1', toName=instName)

#
-----------------------------------------------------------------------------------------------
-------------------------


# Create mesh
-----------------------------------------------------------------------------------------
# COMPLETE THIS SECTION WITH PLANE STRESS TRIANGULAR ELEMENTS
mdb.models['model_2DVoidPlate'].parts['part_Plate'].setMeshControls(elemShape=
    TRI, regions=
mdb.models['model_2DVoidPlate'].parts['part_Plate'].faces[:])
```

```
mdb.models['model_2DVoidPlate'].parts['part_Plate'].seedPart(size=seed_mesh)
mdb.models['model_2DVoidPlate'].parts['part_Plate'].generateMesh()

#
----------------------------------------------------------------------------------------------------
----------


# Create material
----------------------------------------------------------------------------------
# COMPLETE THIS SECTION WITH AN ELASTIC MATERIAL
mdb.models[modelName].Material(name='kryptonite')
mdb.models[modelName].materials['kryptonite'].Elastic(table=((YoungsMod,
PoissonRatio),
    ))
#
----------------------------------------------------------------------------------------------------
----


# Define sets containing all nodes and elements
-------------------------------------------------------------------
mdb.models[modelName].parts[partName].Set(name='set_AllElements',
elements=mdb.models[modelName].parts[partName].elements)

mdb.models[modelName].parts[partName].Set(name='set_AllNodes',
nodes=mdb.models[modelName].parts[partName].nodes)
#
----------------------------------------------------------------------------------------------------
-------------


# Create section, assign to part
---------------------------------------------------------------------------
# COMPLETE THIS SECTION
mdb.models['model_2DVoidPlate'].HomogeneousSolidSection(material='kryptonite',
    name='plateSection', thickness=thickness)
mdb.models['model_2DVoidPlate'].parts['part_Plate'].SectionAssignment(offset=
    0.0, offsetField='', offsetType=MIDDLE_SURFACE, region=Region(
    faces=mdb.models['model_2DVoidPlate'].parts['part_Plate'].faces[:]),
sectionName='plateSection'
    , thicknessAssignment=FROM_SECTION)
#
----------------------------------------------------------------------------------------------------
-----
```

```
# Create arrays and Sets containing node numbers for all faces of plate
----------------------------------------
eps = center_to_center/100.0

mdb.models[modelName].rootAssembly.Set(edges=
    mdb.models[modelName].rootAssembly.instances[instName].edges.findAt(
    ((width_plate/2.0, height_plate/2.0+major_axis_hole+eps, 0.0), )),
name='set_NodesRightEdge')
mdb.models[modelName].rootAssembly.Set(edges=
    mdb.models[modelName].rootAssembly.instances[instName].edges.findAt(
    ((0, height_plate/2.0+major_axis_hole+eps, 0.0), )), name='set_NodesLeftEdge')
mdb.models[modelName].rootAssembly.Set(edges=
    mdb.models[modelName].rootAssembly.instances[instName].edges.findAt(
    ((width_plate/4.0, height_plate, 0.0), )), name='set_NodesTopEdge')
mdb.models[modelName].rootAssembly.Set(edges=
    mdb.models[modelName].rootAssembly.instances[instName].edges.findAt(
    ((width_plate/4.0, 1.*height_plate/2.0, 0.0),)), name='set_NodesBottomEdge')
#
-------------------------------------------------------------------------------------------------
-------------


# Create analysis step
-----------------------------------------------------------------------------
# COMPLETE USING A STATIC STEP
mdb.models[modelName].StaticStep(name='step_Compression', previous='Initial')
#
-------------------------------------------------------------------------------------------------
--------



# Set up BCs
-------------------------------------------------------------------------------------------------
------------
# COMPLETE BY CREATING DISPLACEMENT BCs BIAXIALLY COMPRESSING THE
PART BY 0.005 ON EACH OF THE 4 EDGES
# fix point to prevent rigid body motion
# externally applied strain through the virtual points (y-dir)
#-------------------------------------------------------
a = mdb.models[modelName].rootAssembly
mdb.models[modelName].DisplacementBC(amplitude=UNSET, createStepName=
    'step_Compression', distributionType=UNIFORM, fieldName='', fixed=OFF,
    localCsys=None, name='right', region=a.sets['set_NodesRightEdge']
    , u1=-0.005, u2=0.0, u3=UNSET, ur1=UNSET, ur2=UNSET, ur3=UNSET)
```

```
mdb.models[modelName].DisplacementBC(amplitude=UNSET, createStepName=
    'step_Compression', distributionType=UNIFORM, fieldName='', fixed=OFF,
    localCsys=None, name='left', region=a.sets['set_NodesLeftEdge']
    , u1=0.005, u2=0.0, u3=UNSET, ur1=UNSET, ur2=UNSET, ur3=UNSET)
mdb.models[modelName].DisplacementBC(amplitude=UNSET, createStepName=
    'step_Compression', distributionType=UNIFORM, fieldName='', fixed=OFF,
    localCsys=None, name='top', region=a.sets['set_NodesTopEdge']
    , u1=0.0, u2=-0.005, u3=UNSET, ur1=UNSET, ur2=UNSET, ur3=UNSET)
mdb.models[modelName].DisplacementBC(amplitude=UNSET, createStepName=
    'step_Compression', distributionType=UNIFORM, fieldName='', fixed=OFF,
    localCsys=None, name='bottom', region=a.sets['set_NodesBottomEdge']
    , u1=0.0, u2=0.005, u3=UNSET, ur1=UNSET, ur2=UNSET, ur3=UNSET)
#
----------------------------------------------------------------------------------------------------
----------------------


# Create remeshing rule
----------------------------------------------------------------------------------------------------
-
mdb.models[modelName].rootAssembly.regenerate()
mdb.models[modelName].RemeshingRule(coarseningFactor=DEFAULT_LIMIT,
    description='', maxSolutionErrorTarget=8.0, meshBias=9, minElementSize=
    1e-04, minSolutionErrorTarget=2.0, name='RemeshingRule-1', outputFrequency=
    LAST_INCREMENT, refinementFactor=DEFAULT_LIMIT, region=Region(
    faces=mdb.models[modelName].rootAssembly.instances[instName].faces.findAt(
    ((center_to_center/2.0, 3.0*height_plate/4.0, 0.0), (0.0, 0.0, 1.0)), )), sizingMethod=
    MINIMUM_MAXIMUM, specifyMaxSize=False, specifyMinSize=True, stepName=
    'step_Compression', variables=('ENDENERI','MISESERI', ))
#
----------------------------------------------------------------------------------------------------
----------------------


# Create and submit Job
----------------------------------------------------------------------------------------------------
-
mdb.AdaptivityProcess(job=Job(contactPrint=OFF, description='', echoPrint=OFF,
    historyPrint=OFF, memory=16000, memoryUnits=MEGA_BYTES,
    model=modelName, modelPrint=OFF, multiprocessingMode=DEFAULT,
    nodalOutputPrecision=SINGLE, numCpus=1, numDomains=1, queue=None,
    scratch='', type=ANALYSIS, userSubroutine=''), jobPrefix='', maxIterations=
    5, name='Adaptivity-1')
mdb.adaptivityProcesses['Adaptivity-1'].submit(waitForCompletion=1)
mdb.saveAs(pathName=subPath + 'job_2DVoidPlate.cae')
```

```
#
————————————————————————————————————
———————————————————————

2. Main code for the FEA:
#
-----------------------------------------------------------------------------------------------
----
# German Markaryan
# 11/6/2025
#
# MECH 151/251: Lab 4
# 2D Periodic RVE with Adjustable Hole Shape
#
-----------------------------------------------------------------------------------------------
----

Mdb()
pathName = "Z:/dcengr/My Documents/MECH151L/Lab 4/"
os.chdir(pathName)

# Includes
-------------------------------------------------------------------------------------------
from part import *
from material import *
from section import *
from assembly import *
from step import *
from interaction import *
from load import *
from mesh import *
from optimization import *
from job import *
from sketch import *
from visualization import *
from connectorBehavior import *
session.journalOptions.setValues(replayGeometry=COORDINATE,recoverGeometry=C
OORDINATE)
#
-----------------------------------------------------------------------------------------------
-----

# Rename model
------------------------------------------------------------------------------------------
modelName = 'model_2DVoidPlate'
mdb.models.changeKey(fromName='Model-1', toName=modelName)
```

```
#
--------------------------------------------------------------------------------------------
----


# Create virtual point parts
-----------------------------------------------------------------------------
mdb.models[modelName].Part(dimensionality=TWO_D_PLANAR, name='part_VPx',
type=DEFORMABLE_BODY)
mdb.models[modelName].parts['part_VPx'].ReferencePoint(point=(0.0, 0.0, 0.0))

mdb.models[modelName].Part(dimensionality=TWO_D_PLANAR, name='part_VPy',
type=DEFORMABLE_BODY)
mdb.models[modelName].parts['part_VPy'].ReferencePoint(point=(0.0, 0.0, 0.0))
#
--------------------------------------------------------------------------------------------
----


# Material properties
-------------------------------------------------------------------------------
YoungsMod = 200e3   # Young's modulus (in MPa)
PoissonRatio = 0.3
#
--------------------------------------------------------------------------------------------
----


# Parameters ----------------------------------------------------------------------------
center_to_center = 10       # center to center distance between the holes (in mm)
porosity = 0.03                     # porosity
aspect_ratio = 1.0      # ratio between major and minor axis for the holes
thickness = 0.0                     # thickness of the plates
seed_mesh =0.25                     # seed-mesh (in mm)


width_plate = center_to_center*2.0        # width of plate  COMPLETE
height_plate = center_to_center*2.0       # height of plate COMPLETE
area_plate = height_plate*width_plate# area of the plate COMPLETE
area_hole = porosity*area_plate/4.0 # area of each void COMPLETE
major_axis_hole = sqrt(aspect_ratio*area_hole/pi) # major axis of each void COMPLETE
minor_axis_hole = area_hole/(pi*major_axis_hole) # minor axis of each void COMPLETE


subPath = pathName + "P" + str(int(porosity*100)) + "_AR" + str(aspect_ratio) + "_sS" +
str(int(seed_mesh*100)) + "/"
```

```
if not os.path.exists(subPath):
    os.makedirs(subPath)
os.chdir(subPath)
#
--------------------------------------------------------------------------------------------------------
----

# Create instances for 2 virtual points
-----------------------------------------------------------------
mdb.models[modelName].rootAssembly.DatumCsysByDefault(CARTESIAN)

# virtual point to constrain x direction
mdb.models[modelName].rootAssembly.Instance(dependent=ON, name='inst_VPx',
part=mdb.models[modelName].parts['part_VPx'])

# virtual point to constrain y motion
mdb.models[modelName].rootAssembly.Instance(dependent=ON, name='inst_VPy',
part=mdb.models[modelName].parts['part_VPy'])
#
--------------------------------------------------------------------------------------------------------
-------


# Create mesh
-----------------------------------------------------------------------------------------
# COMPLETE THIS SECTION USING QUADRATIC TRIANGULAR ELEMENTS
# WHEN DEFINING THE SEED, MAKE SURE CURVATURE CONTROL IS OFF
partName='part_Plate'
instName = 'inst_2DVoidPlate'
mdb.models[modelName].PartFromOdb(instance='INST_2DVOIDPLATE', name =
'part_Plate-mesh-1',
    odb=session.openOdb(pathName + "AMR/" + "P" + str(int(porosity*100)) + "_AR" +
str(aspect_ratio) + "/Adaptivity-1-iter2.odb"))

#
--------------------------------------------------------------------------------------------------------
----------


# Create material
-------------------------------------------------------------------------------
# COMPLETE THIS SECTION USING A LINEAR ELASTIC MATERIAL MODEL
mdb.models[modelName].Material(name='kryptonite')
mdb.models[modelName].materials['kryptonite'].Elastic(table=((YoungsMod,
PoissonRatio),
    ))
```

```
#
----------------------------------------------------------------------------------------------------
# Make meshed duplicates of original quarter part
mdb.models[modelName].Part(compressFeatureList=ON, mirrorPlane=YZPLANE,
name='part_Plate-mesh-2',
    objectToCopy=mdb.models[modelName].parts['part_Plate-mesh-1'])
mdb.models[modelName].Part(compressFeatureList=ON, mirrorPlane=XZPLANE,
name='part_Plate-mesh-3',
    objectToCopy=mdb.models[modelName].parts['part_Plate-mesh-1'])
mdb.models[modelName].Part(compressFeatureList=ON, mirrorPlane=XZPLANE,
name='part_Plate-mesh-4',
    objectToCopy=mdb.models[modelName].parts['part_Plate-mesh-2'])
mdb.models[modelName].rootAssembly.regenerate()

mdb.models[modelName].rootAssembly.Instance(dependent=ON, name='part_Plate-
mesh-1-1', part=mdb.
    models[modelName].parts['part_Plate-mesh-1'])
mdb.models[modelName].rootAssembly.Instance(dependent=ON, name='part_Plate-
mesh-1-2', part=mdb.
    models[modelName].parts['part_Plate-mesh-2'])
mdb.models[modelName].rootAssembly.Instance(dependent=ON, name='part_Plate-
mesh-2-1', part=mdb.
    models[modelName].parts['part_Plate-mesh-3'])
mdb.models[modelName].rootAssembly.Instance(dependent=ON, name='part_Plate-
mesh-2-2', part=mdb.
    models[modelName].parts['part_Plate-mesh-4'])

# Translate 4 quarters into position and merge into 1 part ------------------------------  #
CHANGED
mdb.models[modelName].rootAssembly.translate(instanceList=('part_Plate-
mesh-1-2', ),
    vector=(2.0*center_to_center, 0.0, 0.0))
mdb.models[modelName].rootAssembly.translate(instanceList=('part_Plate-
mesh-2-1', ),
    vector=(0.0, 2.0*center_to_center, 0.0))
mdb.models[modelName].rootAssembly.translate(instanceList=('part_Plate-
mesh-2-2', ),
    vector=(2.0*center_to_center, 2.0*center_to_center, 0.0))

mdb.models[modelName].rootAssembly.previewMergeMeshes(instances=(
    mdb.models[modelName].rootAssembly.instances['part_Plate-mesh-1-1'],
    mdb.models[modelName].rootAssembly.instances['part_Plate-mesh-1-2'],
    mdb.models[modelName].rootAssembly.instances['part_Plate-mesh-2-1'],
    mdb.models[modelName].rootAssembly.instances['part_Plate-mesh-2-2']),
    nodeMergingTolerance=1e-06)
```

```
mdb.models[modelName].rootAssembly.PartFromBooleanMerge(domain=MESH,
    instances=(
        mdb.models[modelName].rootAssembly.instances['part_Plate-mesh-1-1'],
        mdb.models[modelName].rootAssembly.instances['part_Plate-mesh-1-2'],
        mdb.models[modelName].rootAssembly.instances['part_Plate-mesh-2-1'],
        mdb.models[modelName].rootAssembly.instances['part_Plate-mesh-2-2']),
    mergeNodes=BOUNDARY_ONLY, name=partName, nodeMergingTolerance=1e-06)
# CHANGED

mdb.models[modelName].rootAssembly.Instance(dependent=ON, name=instName,
    part=mdb.models[modelName].parts[partName])
mdb.models[modelName].rootAssembly.suppressFeatures((
    'part_Plate-mesh-1-1', 'part_Plate-mesh-1-2',
    'part_Plate-mesh-2-1', 'part_Plate-mesh-2-2'))

# Define sets containing all nodes and elements
-----------------------------------------------------------------
mdb.models[modelName].parts[partName].Set(name='set_AllElements',
elements=mdb.models[modelName].parts[partName].elements)

mdb.models[modelName].parts[partName].Set(name='set_AllNodes',
nodes=mdb.models[modelName].parts[partName].nodes)
#
-------------------------------------------------------------------------------------------------
-------------
# Create section, assign to part
-------------------------------------------------------------------------
# COMPLETE THIS SECTION
# Create section, assign to part
mdb.models[modelName].HomogeneousSolidSection(material='kryptonite',
name='section-2DVoidPlate',
    thickness=thickness)
mdb.models[modelName].parts[partName].SectionAssignment(offset=0.0,
offsetField='', offsetType=
    MIDDLE_SURFACE,
region=mdb.models[modelName].parts[partName].sets['set_AllElements'],
sectionName='section-2DVoidPlate')
#
-------------------------------------------------------------------------------------------------
-----
```

```
# Create arrays and Sets containing node numbers for all faces of plate
-----------------------------------------

# initialize arrays for faces (denoted by outward normal)
# nodes_rightEdge = []
# nodes_leftEdge = []
# nodes_topEdge = []
# nodes_bottomEdge = []
# node_RBM = []


# # define arbitrary tolerance for boolean comparison
# eps = 1e-4
# eps2 = 1000.0*eps

# # loop over all nodes and sort out nodes on the edges
# for N in mdb.models[modelName].parts[partName].nodes:

    # nodeCoord = N.coordinates

    # if (fabs(nodeCoord[0]-center_to_center) < eps2) and (fabs(nodeCoord[1]-
minor_axis_hole) < eps2): node_RBM.append(N.label)

    # elif (fabs(nodeCoord[0]) < eps): nodes_leftEdge.append(N.label)

    # elif (fabs(nodeCoord[0]-width_plate) < eps): nodes_rightEdge.append(N.label)

    # elif (fabs(nodeCoord[1]) < eps): nodes_bottomEdge.append(N.label)

    # elif (fabs(nodeCoord[1]-height_plate) < eps): nodes_topEdge.append(N.label)

#
mdb.models[modelName].parts[partName].SetFromNodeLabels(name='set_NodesRigh
tEdge', nodeLabels=nodes_rightEdge)
#
mdb.models[modelName].parts[partName].SetFromNodeLabels(name='set_NodesLeft
Edge', nodeLabels=nodes_leftEdge)
#
mdb.models[modelName].parts[partName].SetFromNodeLabels(name='set_NodesTop
Edge', nodeLabels=nodes_topEdge)
#
mdb.models[modelName].parts[partName].SetFromNodeLabels(name='set_NodesBott
omEdge', nodeLabels=nodes_bottomEdge)
#
mdb.models[modelName].parts[partName].SetFromNodeLabels(name='set_NodeRBM'
, nodeLabels=(node_RBM[0],))
```

```
# # create sets for virtual points
# mdb.models[modelName].parts['part_VPx'].Set(name='set_VPx',
referencePoints=(mdb.models[modelName].parts['part_VPx'].referencePoints[1], ))
# mdb.models[modelName].parts['part_VPy'].Set(name='set_VPy',
referencePoints=(mdb.models[modelName].parts['part_VPy'].referencePoints[1], ))

# #
----------------------------------------------------------------------------------------------------
-------------


# # Create sets of periodic node pairs
--------------------------------------------------------------------------

# # Look at left and right sides
# for i in range (0, len(nodes_leftEdge)):
        # leftCoords =
mdb.models[modelName].parts[partName].sets['set_NodesLeftEdge'].nodes[i].coordin
ates
        #
mdb.models[modelName].parts[partName].SetFromNodeLabels(name='set_NodesLPai
r_' + str(i), nodeLabels=(nodes_leftEdge[i],))
        # for j in range (0, len(nodes_rightEdge)):
                # rightCoords =
mdb.models[modelName].parts[partName].sets['set_NodesRightEdge'].nodes[j].coordi
nates
                # if (fabs(leftCoords[1] - rightCoords[1]) < eps):
                        #
mdb.models[modelName].parts[partName].SetFromNodeLabels(name='set_NodesRPai
r_' + str(i), nodeLabels=(nodes_rightEdge[j],))


# # Look at top and bottom sides
# for i in range (0, len(nodes_topEdge)):
        # topCoords =
mdb.models[modelName].parts[partName].sets['set_NodesTopEdge'].nodes[i].coordina
tes
        #
mdb.models[modelName].parts[partName].SetFromNodeLabels(name='set_NodesTPai
r_' + str(i), nodeLabels=(nodes_topEdge[i],))
        # for j in range (0, len(nodes_bottomEdge)):
                # bottomCoords =
mdb.models[modelName].parts[partName].sets['set_NodesBottomEdge'].nodes[j].coor
dinates
                # if (fabs(topCoords[0] - bottomCoords[0]) < eps):
```

```
                        #
mdb.models[modelName].parts[partName].SetFromNodeLabels(name='set_NodesBPai
r_' + str(i), nodeLabels=(nodes_bottomEdge[j],))

# #
----------------------------------------------------------------------------------------------------
--------------


# # Create analysis step
----------------------------------------------------------------------------
# mdb.models[modelName].StaticStep(name='step_Compression', previous='Initial')
# #
----------------------------------------------------------------------------------------------------
--------


# # Set up BCs
----------------------------------------------------------------------------------------------------
------------

# # fix point to prevent rigid body motion
# mdb.models[modelName].DisplacementBC(amplitude=UNSET, createStepName=
    # 'step_Compression', distributionType=UNIFORM, fieldName='', fixed=OFF,
    # localCsys=None, name='bc_preventRBM', region=
    # mdb.models[modelName].rootAssembly.instances[instName].sets['set_NodeRBM']
    # , u1=0.0, u2=0.0, ur3=UNSET)


# # externally applied strain through the virtual points (x-dir)
# #-------------------------------------------------------------
# mdb.models[modelName].DisplacementBC(amplitude=UNSET, createStepName=
    # 'step_Compression', distributionType=UNIFORM, fieldName='', fixed=OFF,
    # localCsys=None, name='bc_VPx', region=
    # mdb.models[modelName].rootAssembly.instances['inst_VPx'].sets['set_VPx']
    # , u1=UNSET, u2=0.0, u3=UNSET, ur1=UNSET, ur2=UNSET, ur3=UNSET)


# # externally applied strain through the virtual points (y-dir)
# #-------------------------------------------------------------
# mdb.models[modelName].DisplacementBC(amplitude=UNSET, createStepName=
    # 'step_Compression', distributionType=UNIFORM, fieldName='', fixed=OFF,
    # localCsys=None, name='bc_VPy', region=
    # mdb.models[modelName].rootAssembly.instances['inst_VPy'].sets['set_VPy']
    # , u1=0.0, u2=-0.005, u3=UNSET, ur1=UNSET, ur2=UNSET, ur3=UNSET)
```

```
# #
----------------------------------------------------------------------------------------------------
----------------------


# # Set up periodic constraint equations
------------------------------------------------------------------------------

# # right and left edges
# for i in range(0,len(nodes_leftEdge)):

    # # preparation of Coefficients
    # leftCoord=mdb.models[modelName].parts[partName].sets['set_NodesLPair_' +
str(i)].nodes[0].coordinates
    # rightCoord=mdb.models[modelName].parts[partName].sets['set_NodesRPair_' +
str(i)].nodes[0].coordinates

    # coeff1 = -(rightCoord[0]-leftCoord[0])

    # # x-coordinate (Ux_Vpx, H11)
    # mdb.models[modelName].Equation(name='constraint_xLR_' + str(i), terms=(
        # ( 1.0, 'inst_2DVoidPlate.set_NodesRPair_' + str(i), 1),
        # (-1.0, 'inst_2DVoidPlate.set_NodesLPair_' + str(i), 1),
        # (coeff1, 'inst_VPx.set_VPx', 1)))

    # # y-coordinate (Uy_Vpx, H21)
    # mdb.models[modelName].Equation(name='constraint_yLR_' + str(i), terms=(
        # ( 1.0, 'inst_2DVoidPlate.set_NodesRPair_' + str(i), 2),
        # (-1.0, 'inst_2DVoidPlate.set_NodesLPair_' + str(i), 2),
        # (coeff1, 'inst_VPx.set_VPx', 2)))

# # top and bottom edges
# for i in range(0,len(nodes_bottomEdge)):

    # # preparation of Coefficients
    # bottomCoord=mdb.models[modelName].parts[partName].sets['set_NodesBPair_'
+ str(i)].nodes[0].coordinates
    # topCoord=mdb.models[modelName].parts[partName].sets['set_NodesTPair_' +
str(i)].nodes[0].coordinates

    # coeff2 = -(topCoord[1]-bottomCoord[1])

    # # x-coordinate (Ux_Vpy, H12)
    # mdb.models[modelName].Equation(name='constraint_xTB_' + str(i), terms=(
        # ( 1.0, 'inst_2DVoidPlate.set_NodesTPair_' + str(i), 1),
        # (-1.0, 'inst_2DVoidPlate.set_NodesBPair_' + str(i), 1),
```

```
        # (coeff2, 'inst_VPy.set_VPy', 1)))

    # # y-coordinate (Uy_Vpy, H22)
    # mdb.models[modelName].Equation(name='constraint_yTB_' + str(i), terms=(
        # ( 1.0, 'inst_2DVoidPlate.set_NodesTPair_' + str(i), 2),
        # (-1.0, 'inst_2DVoidPlate.set_NodesBPair_' + str(i), 2),
        # (coeff2, 'inst_VPy.set_VPy', 2)))

# #
------------------------------------------------------------------------------------------------------------
----------------------


# # Field Output
------------------------------------------------------------------------------------------------------------
--------
# # Force/Displacement at the Virtual Points
# mdb.models[modelName].FieldOutputRequest(createStepName=
    # 'step_Compression', name='output_VPx', rebar=EXCLUDE, region=
    # mdb.models[modelName].rootAssembly.instances['inst_VPx'].sets['set_VPx']
    # , sectionPoints=DEFAULT, variables=('RF', 'U'))
# mdb.models[modelName].FieldOutputRequest(createStepName=
    # 'step_Compression', name='output_VPy', rebar=EXCLUDE, region=
    # mdb.models[modelName].rootAssembly.instances['inst_VPy'].sets['set_VPy']
    # , sectionPoints=DEFAULT, variables=('RF', 'U'))
# #
------------------------------------------------------------------------------------------------------------
----------------------


# # Create and submit the job for processing
------------------------------------------------------------------------------
# # COMPLETE THIS SECTION
# mdb.Job(atTime=None, contactPrint=OFF, description='', echoPrint=OFF,
    # explicitPrecision=SINGLE, getMemoryFromAnalysis=True, historyPrint=OFF,
    # memory=90, memoryUnits=PERCENTAGE, model=modelName, modelPrint=OFF,
    # multiprocessingMode=DEFAULT, name='auxeticstructure', nodalOutputPrecision=
    # SINGLE, numCpus=1, numGPUs=0, queue=None, resultsFormat=ODB, scratch='',
    # type=ANALYSIS, userSubroutine='', waitHours=0, waitMinutes=0)
# mdb.jobs['auxeticstructure'].submit(consistencyChecking=OFF)
# mdb.jobs['auxeticstructure'].waitForCompletion()
# mdb.saveAs(pathName=subPath + 'auxeticstructure.cae')
# #
------------------------------------------------------------------------------------------------------------
----------------------
```